Gravwell Training Manual: Reference Material for Gravwell On-Site Training and Certification

Contents

Course Overview

Welcome! We're very excited about your interest in this course.

Gravwell exists to provide data analytics to organizations at a predictable and reasonable cost that enables any business unit to gain insights from data. Founded in 2017, Gravwell was created to address some core deficiencies in the market as experienced by our founders. From the very first line of code they knew people needed a tool that was much more efficient on computing resources, could handle binary data natively in order to shrink disk usage and support advanced machine learning, and didn't charge for every ounce of data like some kind of analytics butcher shop.

One incredibly important aspect of data analytics is the ability to ask ad-hoc questions and to perform rapid data exploration. This requires an "ingest first and ask questions later" approach to data ingestion, indexing, and querying. Additionally, data trends over time are incredibly important for identifying cybersecurity incidents, user behavior analytics, and to extract business KPIs and insights. Gravwell needed a structure-on-read time-series database, but a suitable option didn't really exist. In order to accomplish our mission, Gravwell engineers had to start from scratch to create a full-stack analytics platform built with Go. The success of this bold undertaking is what allows Gravwell to scale so incredibly well and to support such a wide variety of use cases.

This training is a deep dive into the Gravwell platform. We will cover the platform architecture, data ingestion strategies, user management and permissions, automation capabilities, cluster management and troubleshooting, and of course, searching. Lots of searching. This document serves as a reference guide and detailed walkthrough for this course.

We really appreciate your interest in Gravwell. We strive to provide amazing customer service and that extends into this training material. If you have any questions, comments, or suggestions, please do send them to us at feedback@gravwell.io. Your opinion is extremely important to us.

Thanks again, and enjoy the training!

Sincerely,

The Gravwell Team

Chapter 1

Architecture Overview

The Architecture Overview chapter is designed to give you a basic understanding of each of the Gravwell components, how Gravwell models data, how data flows through Gravwell, and the different ways in which Gravwell can be deployed. We will examine a few basic deployments, gradually increasing in capacity and complexity. Gravwell is designed to be able to scale to many hundreds of nodes, but it can just as easily run in a container, with all the services coexisting on a minimal system.

1.1 Terminology

Gravwell is a large system with quite a few moving parts. It is important to start by establishing some terminology so that everyone can speak the same language.

Indexer
: Stores data and manages wells.

Webserver
: Serves web interface, controls and coordinates indexers.

Entry
: A single tagged record or data item (line from a log file, Windows event, packet, etc.)

Enumerated Value
: Named data item that is extracted from the raw entry during a search.

Tag
: Human-readable name for a data group. The most basic grouping of data.

Well
: On-disk collection of entries. Every entry ends up in exactly one well, sorted by tag.

Shard
: A slice of data within a well. Each shard contains about 1.5 days of entries.

Ingester
: Program that accepts raw data and packages it as entries for transport to an indexer.

Renderer
: Query component that collects search output and presents results to a human.

Datastore
: Central authority of users and user-owned objects for distributed webservers.

Search Agent
: Monitors and launches automated queries and scripts on behalf of users.

Cluster Deployment
: Multiple Indexers all participating in a single Gravwell instance.

Distributed Webservers
: Multiple webservers sharing the load of GUI interactions and queries, but controlling the same set of indexers.

Load Balancer
: An HTTP reverse proxy which transparently balances load across multiple webservers.

1.2 Gravwell Entries

Gravwell stores all data as *entries*. An entry consists of a piece of data (just an array of bytes), a timestamp, a tag, and a source address. Each of these components deserves a bit of explanation, so we will cover each separately. Entries are stored in an efficient binary format on disk, but a user-friendly representation of an example entry would look something like this:

```
{
    Data: `127.0.0.1 - frank [10/Oct/2000:13:55:36 -0700]
"GET /apache_pb.gif HTTP/1.0" 200 2326 "http://www.example.com/start.html"
"Mozilla/4.08 [en] (Win98; I ;Nav)"`,
    Timestamp: 2000-10-10 13:55:36 -0700 MST,
    Tag: "apache-logs",
    Source: 10.0.2.3,
}
```

1.2.1 "Timestamp" field

The timestamp is meant to indicate the creation time of the *data*. This is typically extracted from the data itself by the ingester, e.g. by parsing out the timestamps on syslog messages. However, some ingesters such as the packet logger will instead set the timestamp to the current time, since the packet was captured "now".

1.2.2 "Data" field

The data field contains the actual raw data of the entry. If log files are being ingested, the data field will typically contain a single line from the log file. When ingesting network packets, the data field will contain a single binary packet.

1.2.3 "Tag" field

The tag field categorizes the entry. Users refer to tags by strings, e.g. "default" or "pcap" or "windows-logs", but under the hood Gravwell assigns each tag string a unique numeric ID for more efficient storage.

1.2.4 "Source" field

The source field indicates where the entry originated. It is an IPv4 or IPv6 address. It is perhaps the most free-form field of the entry, because it can indicate the machine on which the ingester was located, the machine from which the ingester *read* the entry data, or it can be an entirely arbitrary number chosen as a second layer of categorization beyond the tag. Most ingesters provide configuration options for how the source field should be set.

1.3 Data Ingest

A core function of Gravwell is data ingest; *ingesters* take raw data, package it as entries, and transmit those entries to a Gravwell indexer (or indexers) for storage, indexing, and searching. The Gravwell ingest API[1] is open source and so are most of the core ingesters[2]. Ingesters can often feed from multiple data sources at once, and each ingester may upload its entries to a single indexer or multiple indexers.

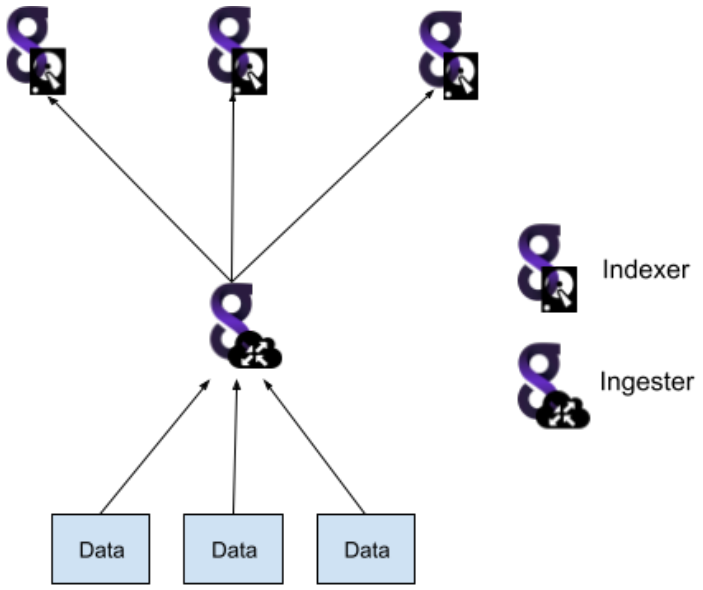


Figure 1.1: Data Ingest

Here are some key concepts for ingesters:

1. Ingesters are separate from indexers.
2. Ingesters may run on remote systems.
3. Ingesters send entries to indexers.
4. Ingesters assign tags to entries.
5. Ingesters can create new tags.
6. Ingesters can cache entries locally if an indexer is not available.
7. Ingesters can multiplex across many indexers.
8. Ingesters can throttle their entry upload rate.
9. Ingesters authenticate with indexers via a secure challenge-response protocol.
10. Authentication is secure over a cleartext connection.
11. Entry transmission is *only* secure when operating over a validated TLS connection with valid TLS certificates.

An entry represents an atomic thing in gravwell whether that be a packet, log line, or binary blob. Entries always contain four fields: timestamp, tag, source, data. The ingesters are responsible for setting those fields.

Entry timestamps are independent of the raw data, although they may be derived from the raw data. For example, an ingester may detect the timestamp in an Apache log line and apply it to the entry, or it may apply the current timestamp.

For more information about available Gravwell ingesters, visit the Gravwell documentation site at https://docs.gravwell.io

[1] https://github.com/gravwell/gravwell/tree/master/ingest
[2] https://github.com/gravwell/gravwell/tree/master/ingesters

1.4 Example Gravwell Deployments

Gravwell can be deployed on a single node, a cluster of indexers and one webserver, a cluster of indexers and a cluster of webservers, or even as geographically-dispersed subclusters. Users interact with Gravwell using either a standards-compliant browser (we like Chromium and Firefox) or the provided Gravwell command line interface. For each deployment example we will show some basic diagrams which will use the icons in Figure 1.2 to depict each component.

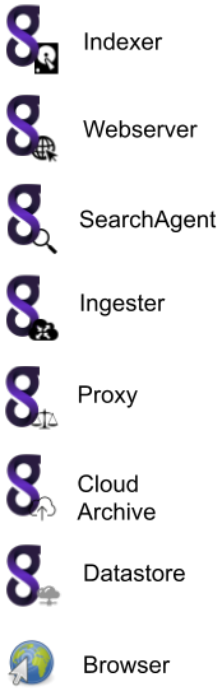


Figure 1.2: Icons used

1.4.1 Single Node

The simplest Gravwell deployment topology is a single node where all components are co-resident on a single host or container. When operating in a single node deployment, Gravwell optimizes queries to maintain data locality. This means that webserver participation in the query pipeline is minimal, typically only running the renderer module. The complete Gravwell deployment contains three core components (Indexer, Webserver, and Search Agent), plus any number of ingesters (both on and off the node) as shown in Figure 1.3.

1.4.2 Cluster Deployment

Gravwell deployments that need to handle large data volumes may require multiple indexers. A Gravwell cluster is comprised of multiple indexers which are controlled by a single webserver as shown in Figure 1.4. The webserver and search agent are typically on a separate node and connect to the indexers via an IPv4 or IPv6 link. When operating in a cluster topology Gravwell will intelligently break up and distribute a query so that as much of the query pipeline as possible runs on the indexers, condensing the pipeline to the webserver only when necessary. Depending on the query, cluster topologies may transmit significant data volumes between the indexers and webserver, so it is important that the links between the webservers and indexers be fast, with low latencies and high bandwidth. While it is possible to geographically separate a webserver from an indexer over a WAN link, the increased latency and decreased bandwidth will affect query performance.

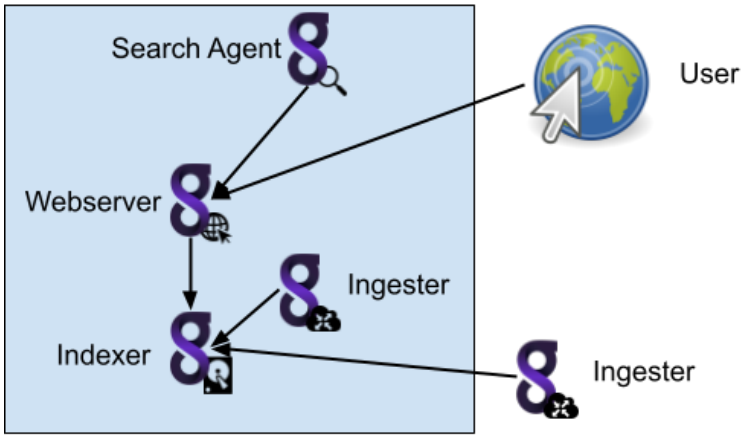


Figure 1.3: Single-node deployment

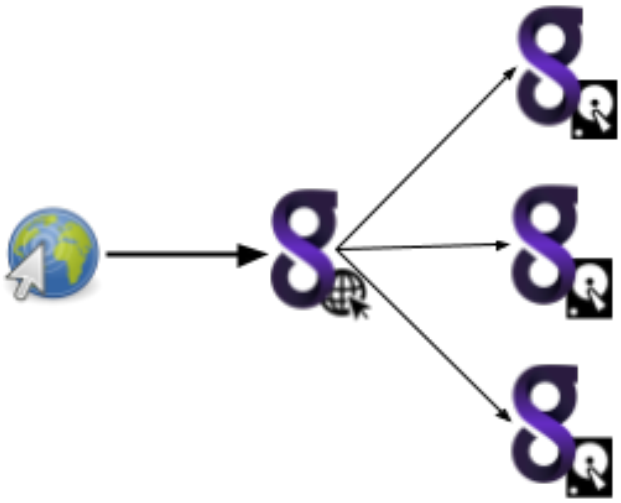


Figure 1.4: Cluster Deployment

1.4.3 Distributed Webserver Architecture

Very large Gravwell deployments with many users may need to also employ multiple webservers to handle the load. Gravwell supports a distributed webserver topology by coordinating and synchronizing the webservers, shown in Figure 1.5.

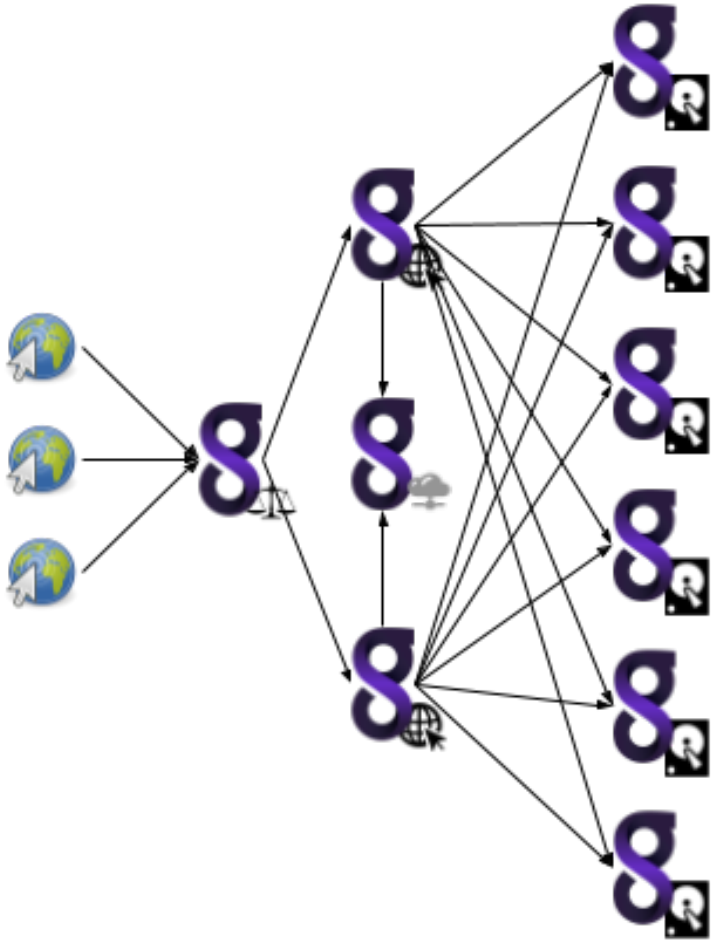


Figure 1.5: Distributed Webserver Architecture

Webservers coordinate and synchronize using a component called the *datastore*. The datastore acts as a master storage system for user data, search resources, scheduled queries, scripts, and any other component that can be uploaded or modified by a user. Webservers and the datastore will attempt to maintain a complete copy of all resources, which allows webservers to fail and be restored by simply rejoining the cluster. However, the datastore is treated as the master copy, so while the datastore component requires minimal CPU resources, robust storage is advised.

1.5 Replication

Gravwell supports full data replication, so that in the event of hardware failure, data is not lost. Replication strategies depend on the type of deployment and general tolerance for distributed failures. Replication is controlled entirely by the indexers and is unaffected by the webserver deployment. Gravwell supports two forms of replication: online and offline.

1.5.1 Online Replication

Online replication requires that indexers communicate directly with one another and coordinate data replication. Online replication allows for hot-failover, meaning that if an indexer fails the other indexers in the replication group will detect the failure and serve the failed node's data during a query.

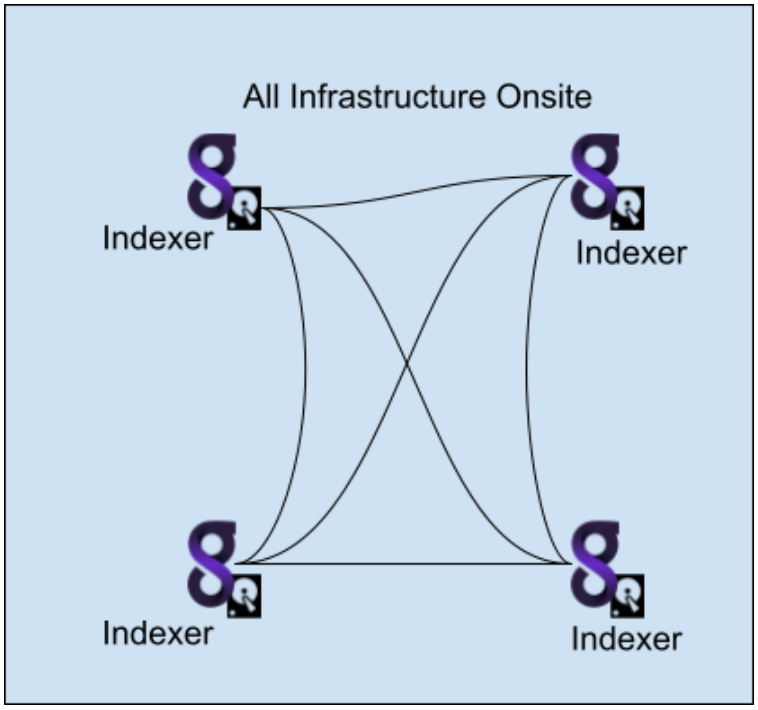


Figure 1.6: Online Replication

1.5.2 Offline Replication

Offline replication is achieved using a dedicated replication component which does not participate in queries. Indexers will replicate their data to the offline replication service and only pull data back in the event of a failure. A single offline replicator may service many indexers (provided that it has adequate storage). The offline replicator can be onsite or offsite.

1.6 Scheduled Search and Orchestration

Gravwell includes an automated search and scripting system. Using scheduled searches and scripts you can keep resources up to date, export data, run multiple queries and alert on the results, or even take action on remote systems based on the results of a query. The scheduled search and orchestration scripts are executed by the search agent. The search agent is a standalone component which communicates with one or more webservers and invokes queries and scripts as needed, as shown in Figure 1.8. There is only ever one search agent in a deployment, and the default Gravwell deployment colocates the searchagent with the webserver. However, in a distributed webserver topology the search agent may be located on its own node.

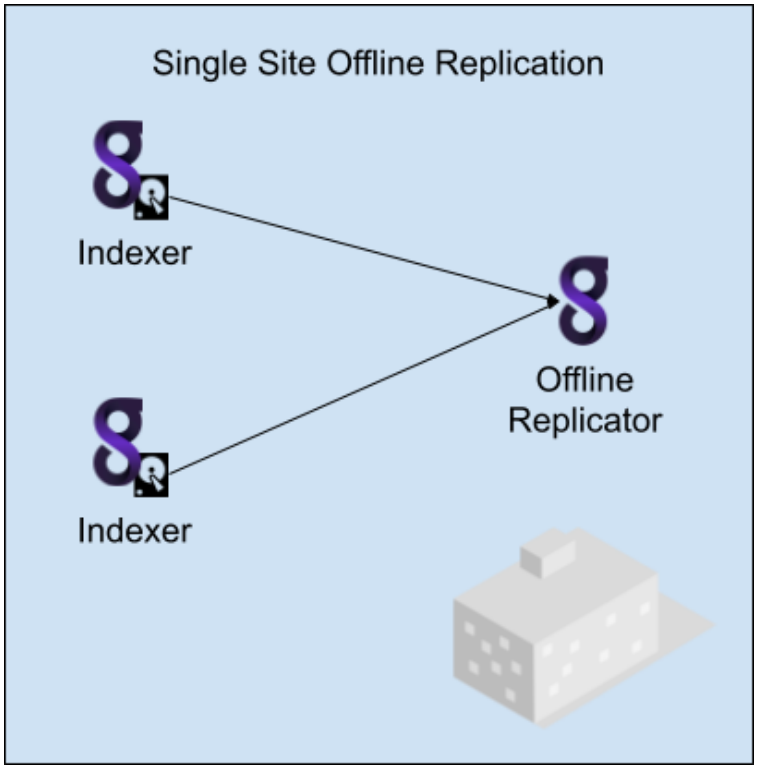


Figure 1.7: Offline Replication

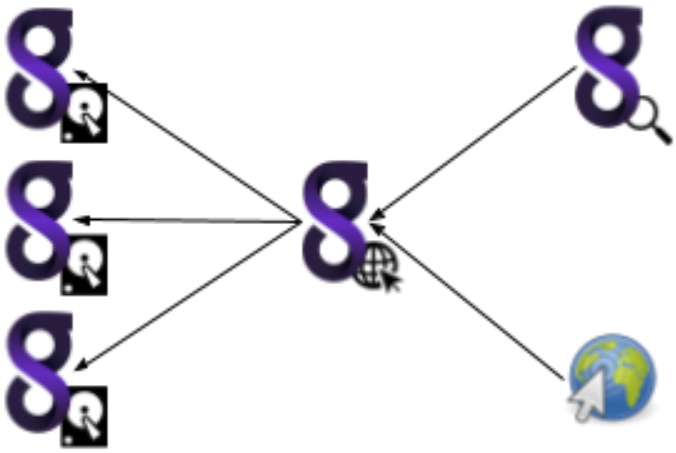


Figure 1.8: Scheduled Search and Orchestration

Chapter 2

Lab Setup and Docker Testing

We will be making extensive use of the Docker container platform during the hands on exercises. The purpose of this chapter is to ensure that all training attendees have a Linux system with a functioning Docker installation. To fully participate in the training each attendee will need the following:

1. Workstation running AMD64 Linux or a virtual machine running 64-bit Linux
   (a) At least 4GB of available RAM
   (b) At least 10GB disk storage available for Docker instances
2. Ethernet adapter and cable with connectivity to training server
3. Standards-compliant web browser (Firefox, Chrome, Chromium)

2.1 Getting Started With Docker

The hands on training segments make heavy use of Docker[1]. Docker is a container platform designed to simplify the deployment and management of microservices. While Gravwell is not a microservice, Docker provides an easy to use and clean environment to experiment with Gravwell services. This section will ensure that you have a working Docker environment and can import Gravwell images.

2.1.1 Host System Requirements

The first thing to verify is that the system that will be hosting Docker and our gravwell images is a modern 64bit Linux kernel. We need at least kernel version 3.2 running on an x86 architecture with 64bit extensions. Any X86 AMD or Intel processor manufactured within the last 10 years should work. Open a command prompt and run the `uname -a` command. We want to see `x86_64` and a kernel version greater than 3.2.0:

```
Linux training 4.18.0-16-generic #17-Ubuntu SMP Fri Feb 8 00:06:57 UTC 2019 x86_64 x86_64
x86_64 GNU/Linux
```

2.1.2 Verifying Docker Installation

Docker is available in most Linux package managers on modern Linux distributions. Most Debian based distributions can install Docker via the command `apt install docker.io` and Redhat based distributions can install docker using `yum install docker`. Make sure to execute these commands as the super user, either via `sudo` or by first becoming root using `su`.

[1] https://www.docker.com

2.1.3 Granting User Access to Docker

Docker requires elevated privileges to start containers, build networks, and generally manage the Docker process. If you don't mind interacting with Docker as the superuser you can skip this section, but if you would like to be able to run the labs as a non-privileged user (recommended) we will set up the `docker` group and add your current user to it. This will allow your current user to drive Docker.

First check if your user can run Docker commands by executing `docker ps -q` and `docker network ls` as a non-root user. If the output is empty or contains a list of running Docker containers, your user already has access to the Docker API. If errors were displayed, we will need to create the docker group and add your current user to it. This is done using the `groupadd` and `usermod` commands. First run `groupadd docker` as the superuser, then run `usermod -aG docker <username>` where `<username>` is the user you wish to use for the training labs. You will need to logout of the current user and then log back in so that the group permission changes can propagate. You can also just reboot your machine/VM. If all goes well, you should be able to run the docker commands as a non-root user.

2.2 Testing Docker Image Imports

A core component of the hands-on training labs is importing prebuilt Gravwell images for experimentation. We need to walk through the basic Docker commands to import and/or load a Docker image and run it. You should have been provided with a `tar.gz` archive containing materials for the course. Make sure it's unpacked, then look in the `dockerimages` subdirectory for a file named "`test.tar.gz`". Load it into your local docker image repository using the command `docker load -i test.tar.gz`, then verify that the image was imported using the `docker images` command. You should see a new image tagged (named) `gravwell:test`.

Now we can start up the test container using Docker and poke around a little. A container is like a very lightweight virtual machine. Note that a container is not quite the same as a Type 1 or Type 2 virtual machine, since it uses the same kernel as the host system.[2].

2.3 Starting a Container in Docker

Now that we have imported a Docker image, let's create a new container and poke around in it a bit. We are going to start the `gravwell:test` image and execute the `/bin/sh` command in interactive mode. This means that we will immediately drop into the container. To keep things clean we will also pass the `--rm` flag which tells Docker to destroy the container when the main process (`/bin/sh`) exits. Start the container and poke around a bit, check the IPs, proc filesystem, and running processes. The container *almost* looks like a real machine.

```
user@training:~$ docker run -it --rm gravwell:test /bin/sh
# ps
PID   USER     TIME  COMMAND
    1 root      0:00 /bin/sh
    6 root      0:00 ps
# ip addr
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc
noqueue qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
       valid_lft forever preferred_lft forever
7: eth0@if8:
<BROADCAST,MULTICAST,UP,LOWER_UP,M-DOWN qdisc
noqueue
```

[2] https://www.backblaze.com/blog/vm-vs-containers/

```
    link/ether 02:42:ac:11:00:02 brd ff:ff:ff:ff:ff:ff
    inet 172.17.0.2/16 brd 172.17.255.255 scope global eth0
       valid_lft forever preferred_lft forever
# whoami
root
# pwd
/
# exit
user@training:~$
```

Type exit to leave the container, and run `docker ps -a` to list all active containers. Because we gave the `--rm` flag the container should be completely gone.

WARNING: Do not use the `--rm` flag on containers that you want to keep or that you expect to continue to work on; one accidental `exit` command will completely destroy all your work.

2.4 Testing Gravwell in a Container

Let's fire up a minimal Gravwell instance in a docker container and verify that we can get to the web portal. A minimal Gravwell instance is available in the images directory at `dockerimages/base.tar.gz` within the training directory. Load it into your local images repository, then run it without any arguments. The base image already has configuration parameters which specify which processes should be started within the container. We will be naming this container `base` using the `--name` parameter and also starting the container in the background using the `-d` parameter.

```
~/gravwell_training$ docker load -i dockerimages/base.tar.gz
Loaded image: gravwell:base
~/gravwell_training$ docker run -d --name test gravwell:base
7c18342e7dcc4e362323797954bd1b1742dd903ded097331f5c92e13853bd628
~/gravwell_training$ docker ps -q
7c18342e7dcc
```

Verify that the container is running using docker ps, then let's pull out the IP that Docker assigned our container so that we can hit the Gravwell webserver.

```
docker inspect test
```

Visit the IP address in your browser; you should see the Gravwell web page. Log into Gravwell with the credentials admin/changeme to make sure everything is OK, then go back to your command prompt and stop and clean up the base image.

```
docker stop test
docker rm test
```

2.5 Loading the Lab Images

Several different Docker container images are used for the lab sections of this training. They are available in the `dockerimages/` subdirectory on the course materials bundle and can be installed with the following command:

```
docker load -i imagename.tar.gz
```

Fetch and load the following images from the dockerimagesimages folder:

- base.tar.gz
- brokenperms.tar.gz
- datastore.tar.gz

- indexer.tar.gz
- ingesters.tar.gz
- offlinereplication.tar.gz
- pcap.tar.gz
- webserver.tar.gz

Once loaded, they should all be in the listing when you run `docker image ls`.

2.6 Creating the Gravwell Test Network

To help keep our Gravwell experimental containers separate, we put them all on a Docker network called "gravnet". This is also what allows containers to connect to each other by name rather than by IP, which is an important part of our experimental config. Before you use the network, you must create it:

```
docker network create gravnet
```

2.7 Helpful Docker Tips

This section has some commands or scripts to help you when doing this training.

To show all IPs and names:

```
docker ps -q | xargs -n 1 docker inspect --format \
'{{range .NetworkSettings.Networks}}{{.IPAddress}}{{end}} {{ .Name }}' \
| sed 's/ \// /'
```

To kill all running containers:

```
docker kill $(docker ps -q)
```

Chapter 3

Using the GUI

3.1 Introduction

This chapter discusses the basic organization and usage of the Gravwell GUI. It begins with a general overview of the user interface, then discusses more specialized and advanced options. Feel free to skip ahead to the next chapter to begin searching with Gravwell immediately, referring back to this chapter when needed.

3.2 Menus

By default, users will end up on the "Welcome" page after logging in. This page, like all pages in the Gravwell GUI, includes a bar across the top containing the main menu, notifications, the account menu, and more. These menus are described further below.

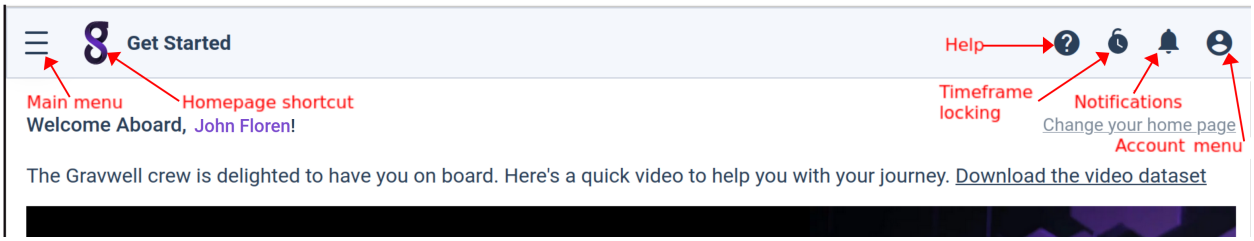


Figure 3.1: Default home page

Clicking the Gravwell logo in the upper left will always take you back to your home page (see section 3.2.3 for configuration options).

3.2.1 The Main Menu

Clicking the "hamburger" button in the upper left will open the Main Menu, as shown in Figure 3.2.

This menu is used to access all the primary functionalities of Gravwell, including dashboards, the query library, and playbooks. Note that several items within the menu are actually sub-menus, which can be expanded to show additional options as shown in Figure 3.3. Items within these sub-menus will typically be used less frequently that the top-level items and are therefore collapsed to save space.

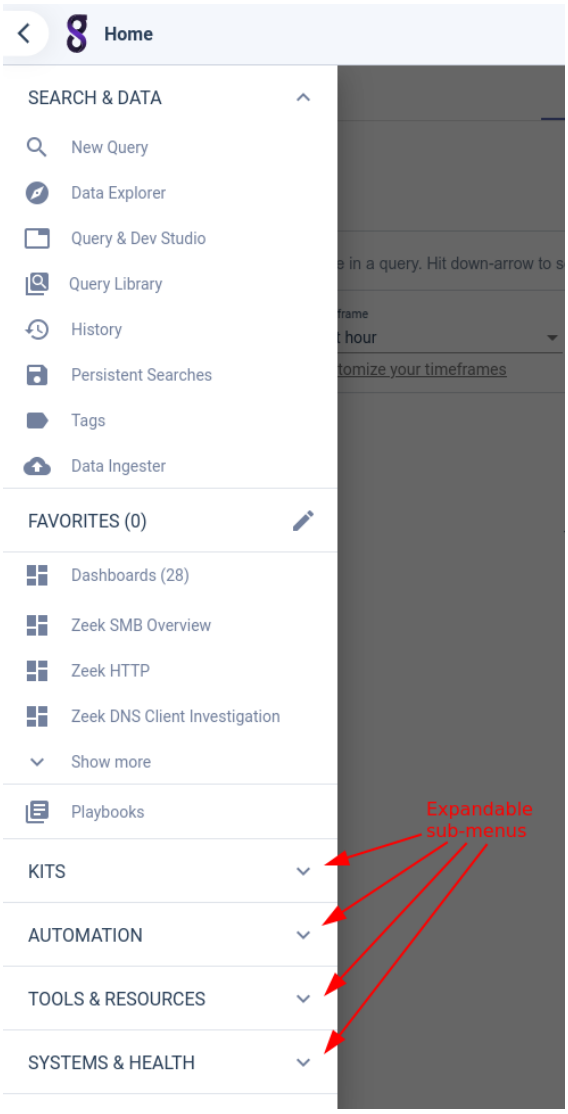


Figure 3.2: The main menu

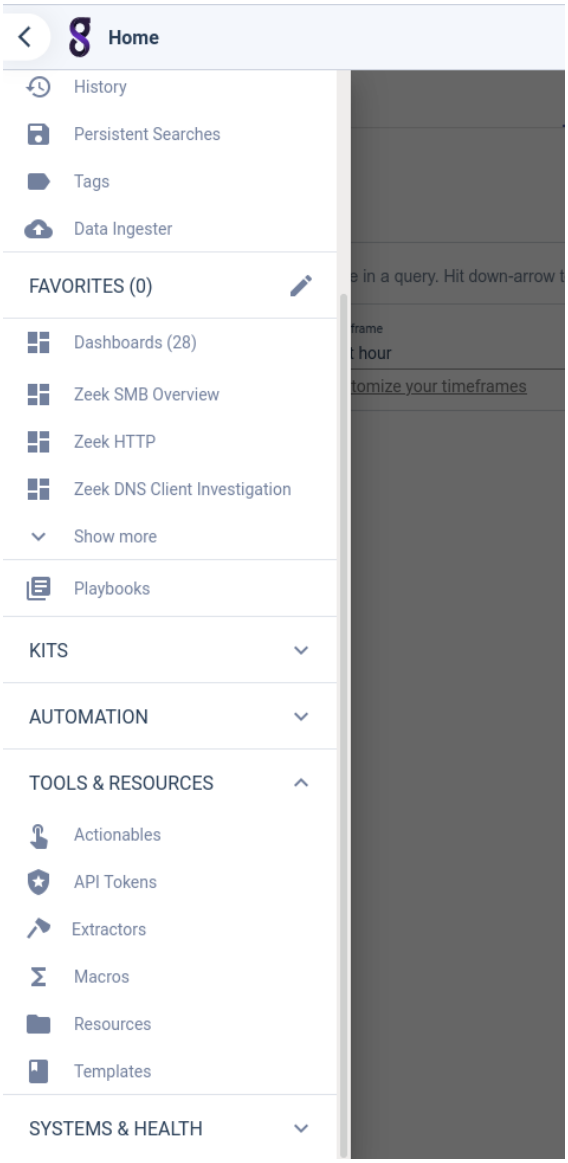


Figure 3.3: Expanded submenu

3.2.2 Notifications

Important notifications are accessible under the bell icon in the upper right corner of the page. Regular notifications are indicated by a small red circle containing the number of notifications. A critical notification will change the entire icon to a more attention-catching red icon; see Figure 3.4 for examples.

Figure 3.4: Notification icons, normal (left) and critical (right)

Clicking the notification icon will display the text of the notifications, as seen in Figure 3.5. Clicking the "snooze" button on a notification will remove that notification from counter shown on the icon; this can be useful to prevent distractions.

Depending on the type of notification, clicking the "delete" icon may clear the notification entirely. Some notifications are persistent and cannot be deleted; some are system-wide and can only be deleted by the administrator, and some are targeted at the current user and can be deleted by that user. Note that there is no harm in clicking "delete" on a notification the user isn't allowed to delete.

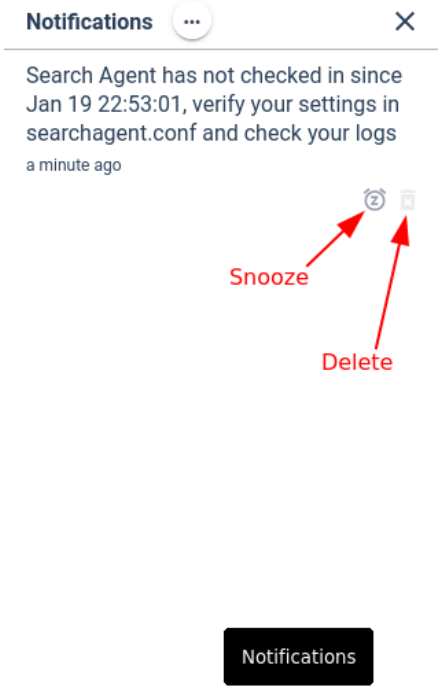


Figure 3.5: Notifications menu

3.2.3 The Account Menu

The round button in the upper-right of the page is the Account Menu button. It will display either the initials of the current user, or a profile image if set. Clicking it brings up a small drop-down menu (Figure 3.6).

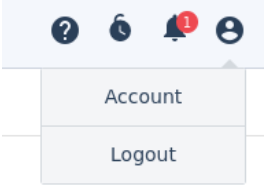


Figure 3.6: The account menu

Selecting "Account" will open your preferences page, shown in Figure 3.7. Here, you can change your email address, display name, or password; be sure to click "Update Account" after making changes! The "Log out all sessions" button at the bottom of the screen will kick *all* active sessions for your account, across all client machines.

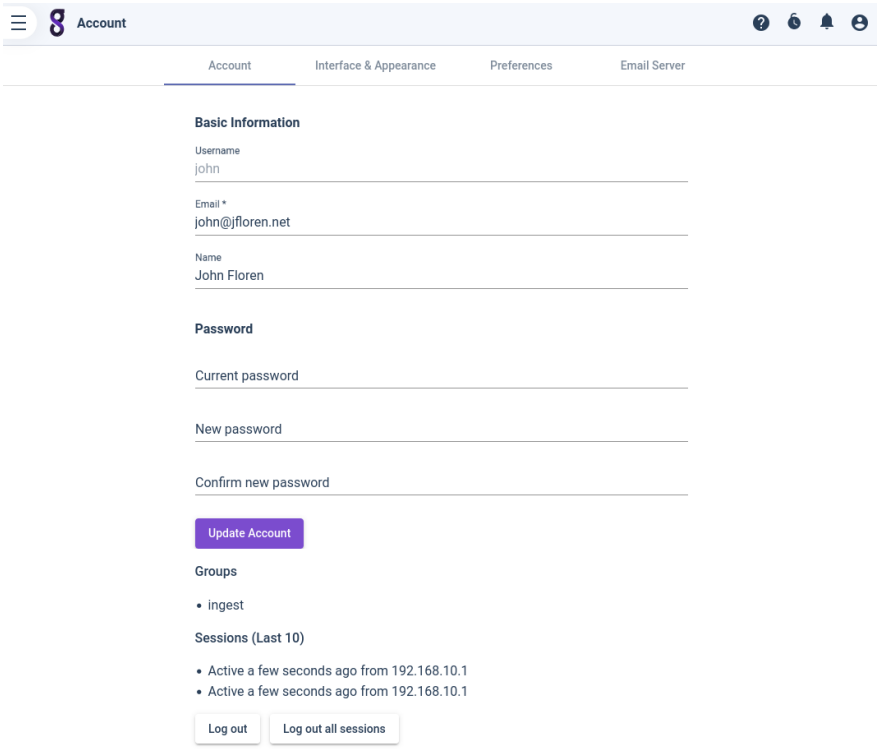


Figure 3.7: Account preferences

The second tab of the Preferences page, "Interface & Appearance", is shown in Figure 3.8. It has options for customizing the Gravwell user interface. The "Interface theme" dropdown is of particular interest, as it selects a GUI-wide color scheme (including the ever-popular dark modes).

The "Chart theme" dropdown selects different color palettes which will be used when drawing charts. The editor theme & font size options control the appearance of Gravwell's built-in text editor, which is used to create automation scripts and in a few other places.

The third tab, "Preferences" (Figure 3.9), allows you to change some default behaviors of Gravwell.

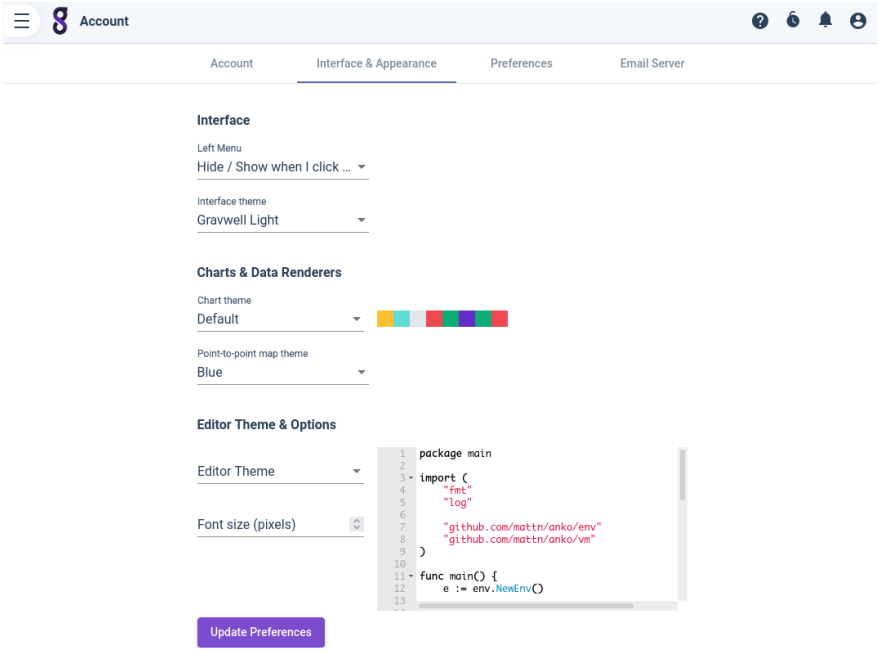


Figure 3.8: Interface and appearance preferences

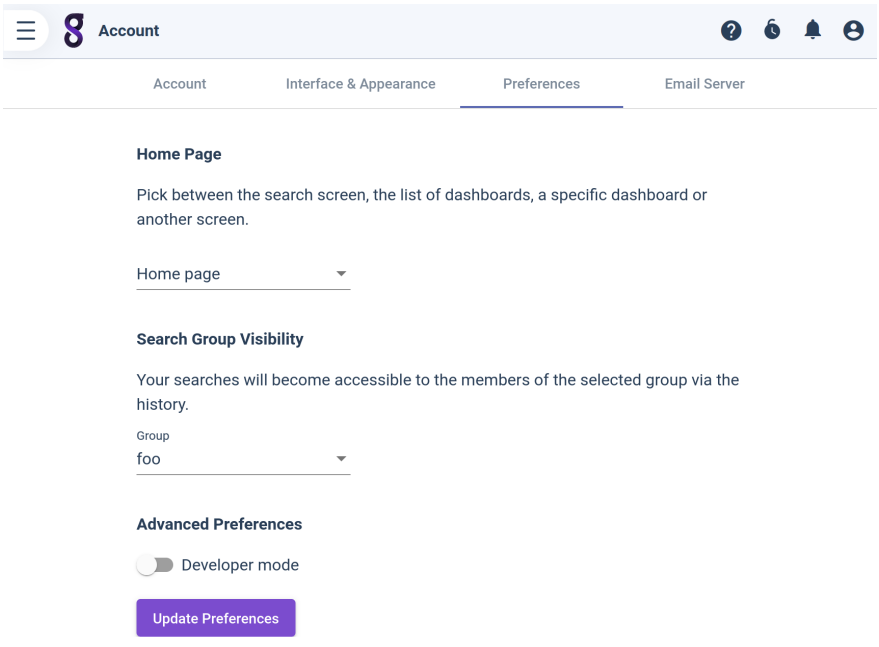


Figure 3.9: General preferences

The “Home Page” dropdown menu selects which page will be displayed after logging in or clicking the Gravwell icon next to the main menu. By default, the welcome page is shown, but you can chose to be shown a list of dashboards, kits, or playbooks instead.

The “Search Group Visibility” option allows you to share the results of all searches with a given group; this can be a convenient way to collaborate. In the screenshot, the user has selected the group named “foo”; all members of that group will have access to the searches this user runs in the future.

The “Advanced Preferences” section can be ignored by most users. Selecting “Developer mode” enables manual editing of JSON preferences.

The final tab, “Email Server” (Figure 3.10, is extremely important for users who intend to do automated email alerting via scheduled scripts. It must be set up with a valid SMTP configuration before emails can be sent.

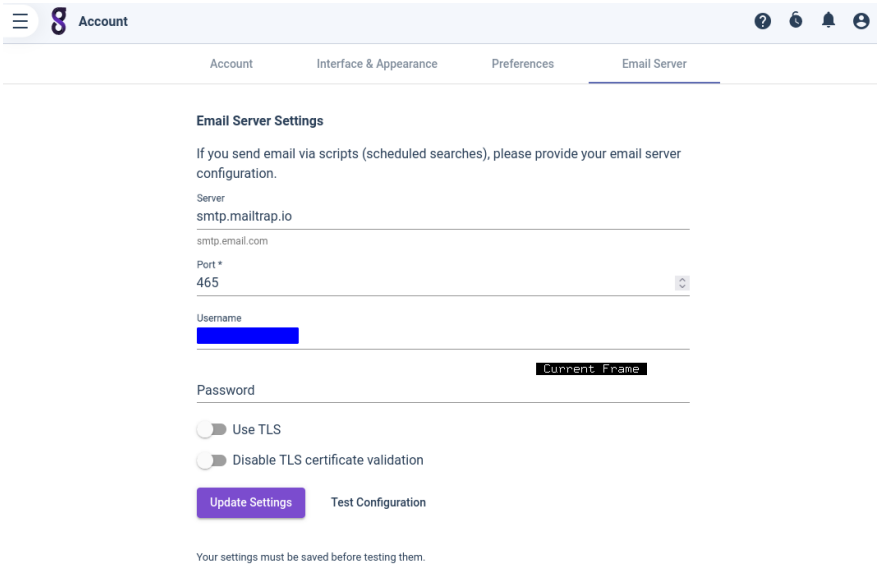


Figure 3.10: Email preferences

The fields are mostly self-explanatory; “Server” is an SMTP server, “Port” is the port to use for SMTP, “Username” and “Password” authenticate to that server. “Use TLS” should be enabled if the server expects TLS connections. The “Disable TLS certification validation” option is provided in case the server is using self-signed certificates; be cautious enabling this!

Once the fields have been populated, click “Update Settings” to save them, then click “Test Configuration” to send a test email.

3.3 Labels and Filtering

Objects in Gravwell such as dashboards, resources, macros, etc. can be labeled for organizational purposes. Some objects distributed in kits may be pre-labeled for convenience. The following object types can be labeled:

- Extractors
- Dashboards
- Kits
- Playbooks
- Resources
- Scheduled Searches / Automation Scripts
- Macros
- Templates
- Actionables
- Query Library entries
- User files

3.3.1 Defining and Managing Labels

Labels are added or deleted in the edit dialog for a given object. While the exact layout of the dialog varies for each object type, they will all have a section for Labels. Labels are added by typing the label into the text bar and hitting enter. Multiple labels can be added in succession in this manner. In Figure 3.11, the user has added three labels (`network`, `asn`, and `lookup`) to a resource.

Labels may be removed in the edit dialog by clicking the "x" next to the label, or by using the backspace key.

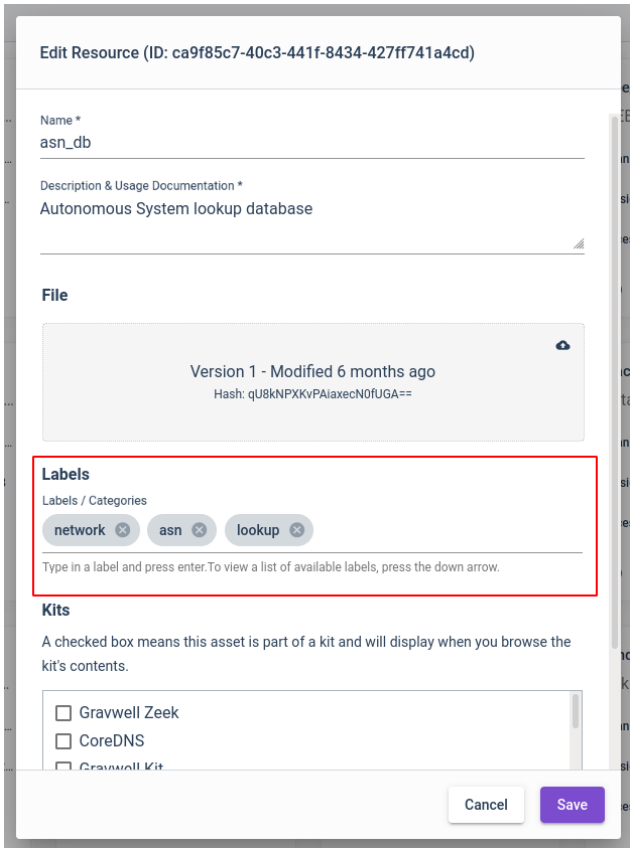


Figure 3.11: Labels applied to a resource

3.3.2 Filtering Objects

The GUI can filter objects based on their labels. At the top of many screens is a bar containing a "Filters" button, as shown in Figure 3.12. Clicking that button brings up a menu with several options for filtering (Fig 3.13).

Multiple filters can be applied simultaneously. When a filter has been applied to a page, a blue "X" icon will appear on the Filters button. Clicking this icon will clear all filters.

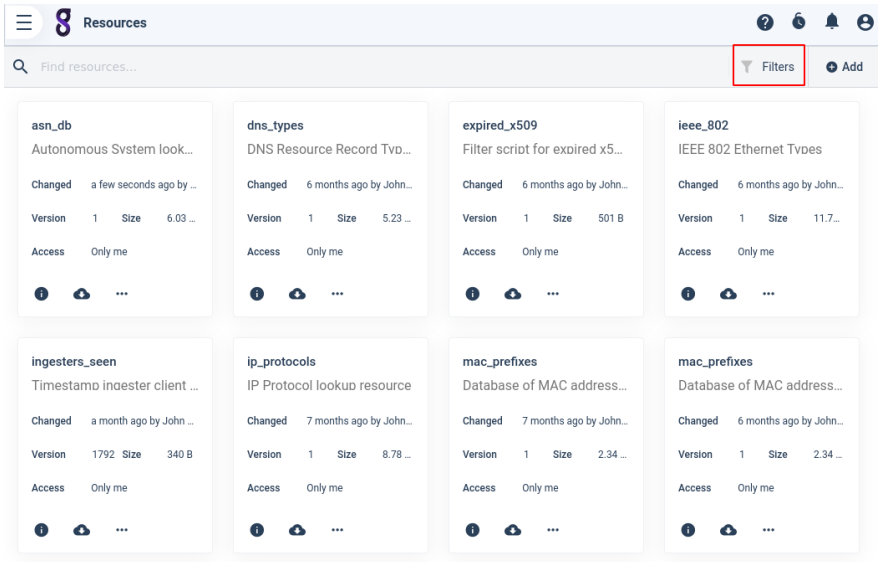


Figure 3.12: The filters menu

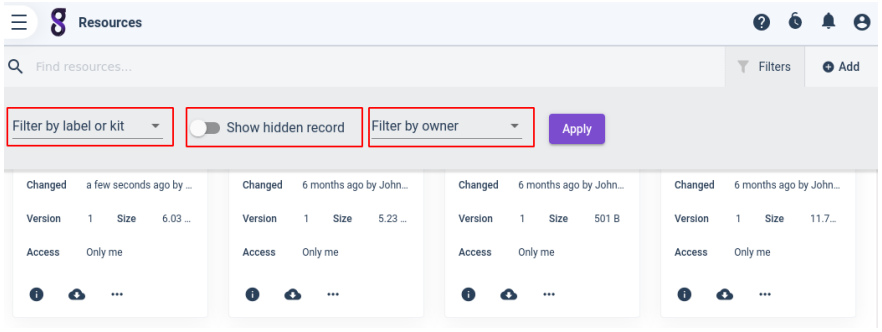


Figure 3.13: Options for filtering

Filtering by Label

The left-most dropdown, "Filter by label or kit", allows you to select one or more labels or kits. Clicking the Apply button will then show only those objects with the specified labels or installed by the specified kits. Figure 3.14 shows the user selecting the "lookup" label, which will hide all resources not labeled "lookup".

Filtering by Owner

By default, most interfaces will show all objects the user has access to, regardless of owner. Another filter option can make it show only objects owned by a particular set of users. Figure 3.15 shows the user filtering macros to show only those macros owned by the user named "Admin User". Note that the "My data" option

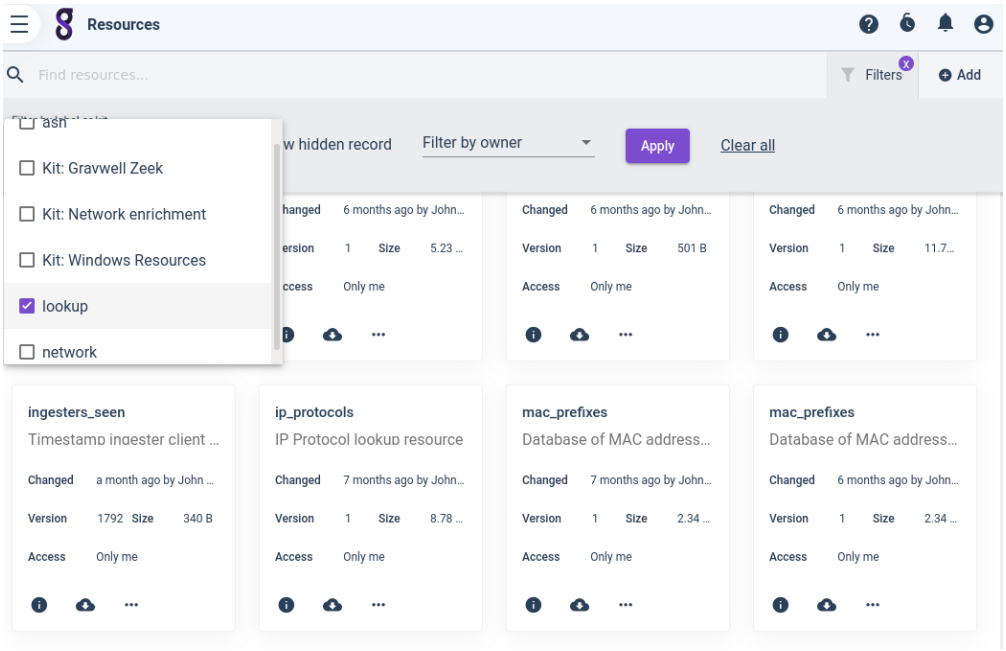


Figure 3.14: Filtering by label

applies to the current user; selecting this checkbox will show only those items which *belong* to the current user, rather than all items the user *can access*.

Showing Hidden Objects

Objects with the special label `hidden` are not displayed by default. This is a convenience function which can keep displays clear of rarely-accessed objects. Click the "Show hidden record" toggle to show hidden objects. In Figures 3.16 and 3.17, toggling "Show hidden record" reveals an additional hidden macro.

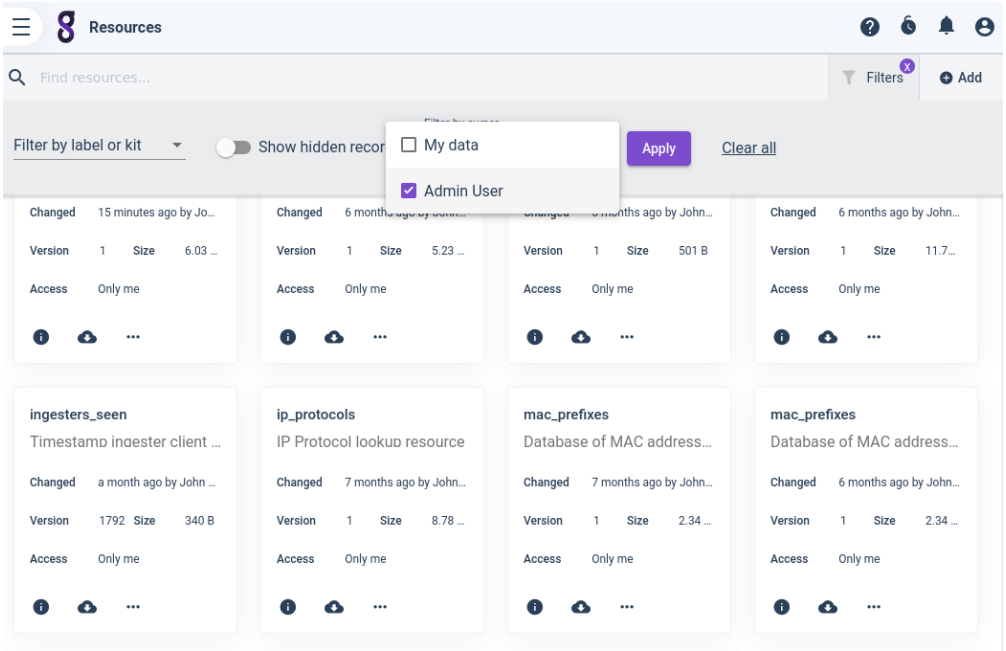


Figure 3.15: Filtering by owner

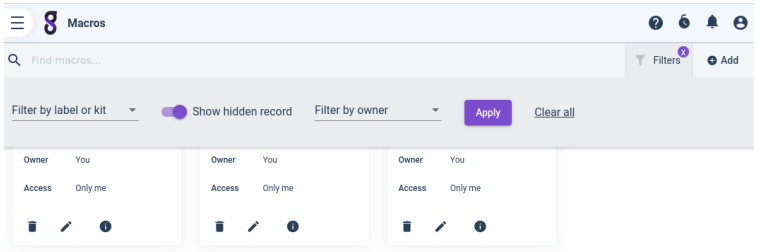


Figure 3.16: Enabling the “show hidden” toggle

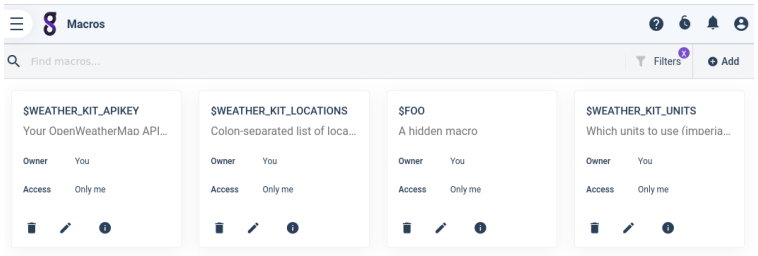


Figure 3.17: Hidden item becomes visible with “show hidden” enabled

3.3.3 Special Labels

Gravwell defines a handful of special labels and label prefixes for particular operations.

The string `hidden` is a special label; applying it to an object will prevent the object from being displayed by default. To see the object, toggle the "Show hidden record" option in the filter menu, as detailed above.

Kit Label Prefixes

Three label prefixes are used to manage Gravwell-internal information about objects which were installed as part of a kit. You should never manually apply kit labels to objects; these labels are documented to prevent users from accidentally applying a conflicting label to an object. The following are considered reserved kit label prefixes:

- kit/
- kit/dependency:
- kit/configuration:

Users should not create labels beginning with these strings, e.g. `kit/foo` or `kit/dependency:bar`. These labels are managed internally by Gravwell.

3.4 Search Interfaces

There are two different user interfaces for running searches in Gravwell: the classic lightweight search interface, and the multi-tabbed Query Studio. Which should you use? Either is acceptable, both take exactly the same Gravwell queries, but most users will find the Query Studio most comfortable.

Throughout this text, screenshots may use either interface interchangeably, because they are *designed* to be used interchangeably. Every feature available in the classic interface is available in the Query Studio. The Query Studio provides a few extra conveniences (including more advanced Data Explorer functionality (Section 4.6), which is why we recommend it to most users.

The classic search interface is accessed via the "New Query" option in the main menu. It presents a very sparse and straightforward way to enter a Gravwell query, along with a few options such as timeframe selection (Figure 3.18).

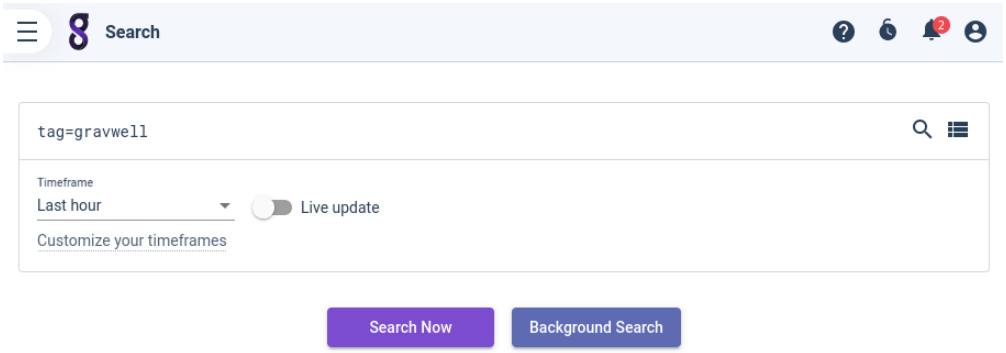


Figure 3.18: The classic search interface

Clicking "Search Now" launches the search, taking the user to the classic results page as seen in Figure 3.19. From here, the query can be modified and iterated on, re-running the search again and again as you explore data.

The Query Studio is a newer interface, designed to provide a convenient multi-tabbed environment in which a user can work with multiple queries at the same time. Figure 3.20 shows what the Query Studio looks like when first opened. Note how it presents a list of queries from the query library, templates, and selections from recent query history; clicking any of these will launch the selected query. Clicking "Start a New Query" begins with a blank query instead.

Figure 3.21 shows the Query Studio with two tabs open. The current tab displays the results of a chart query. Note the "Pie Chart" text near the upper right; this is a drop-down menu which allows the user to quickly change between chart types when running a query with the chart renderer. This and other conveniences make the Query Studio a more comfortable environment for most users.

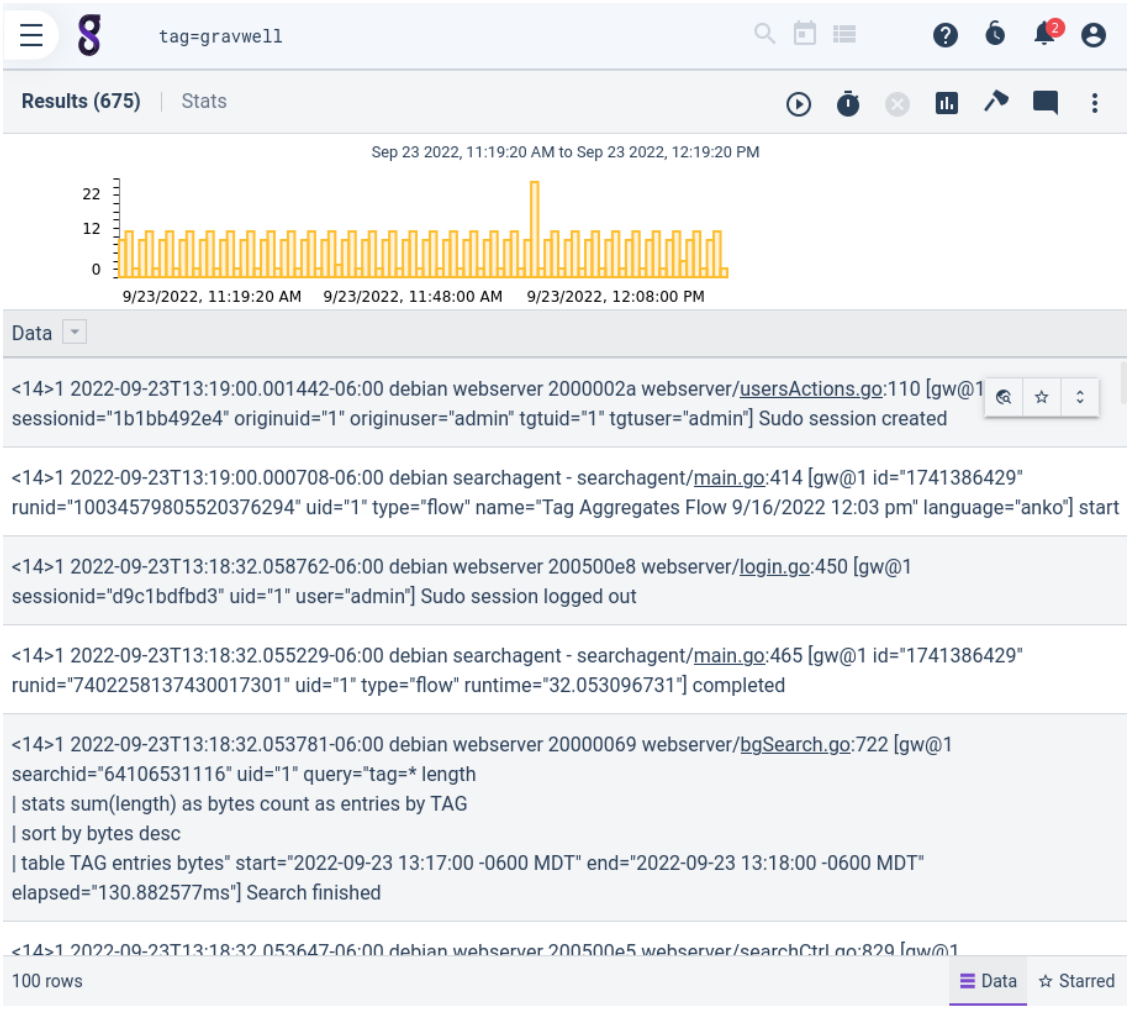


Figure 3.19: The classic search results page

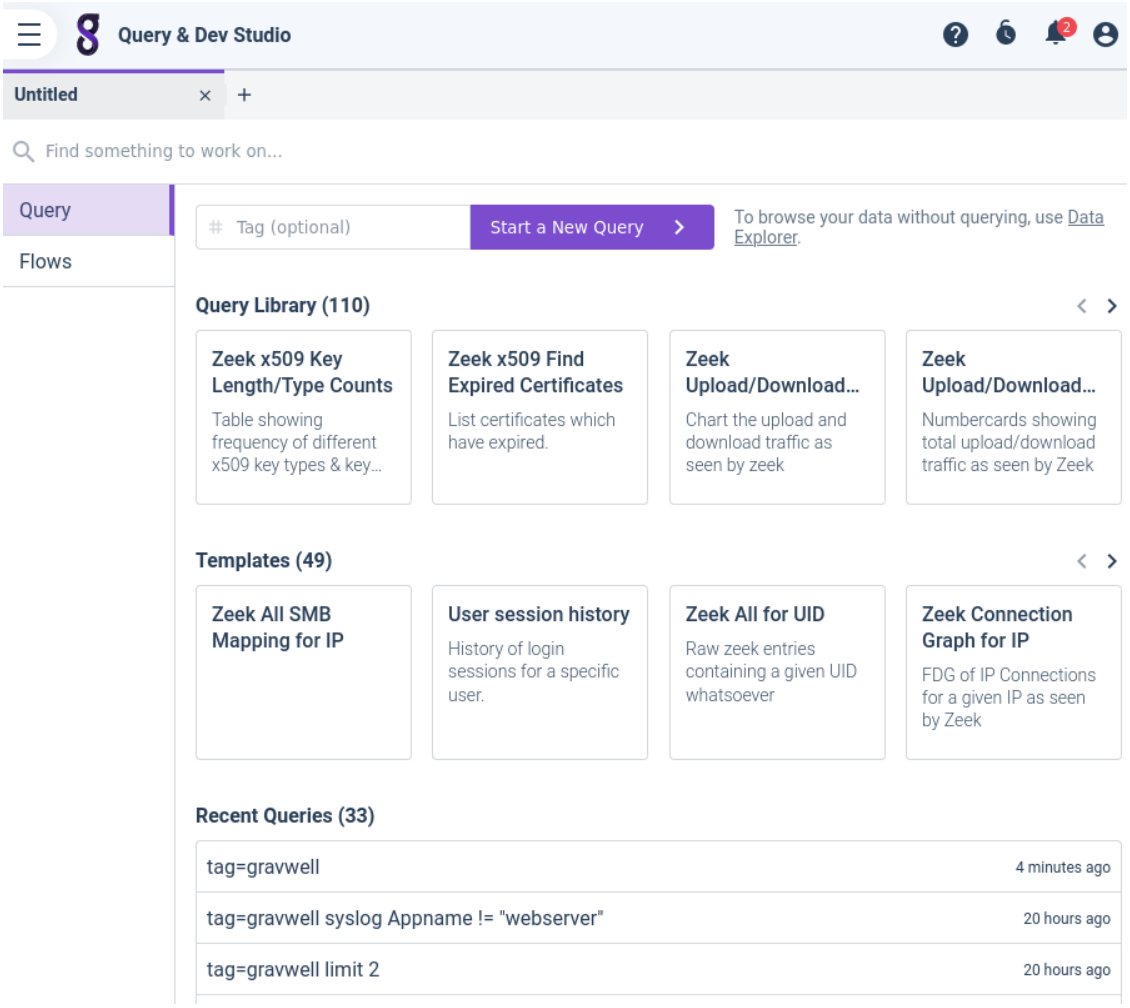


Figure 3.20: The Query Studio

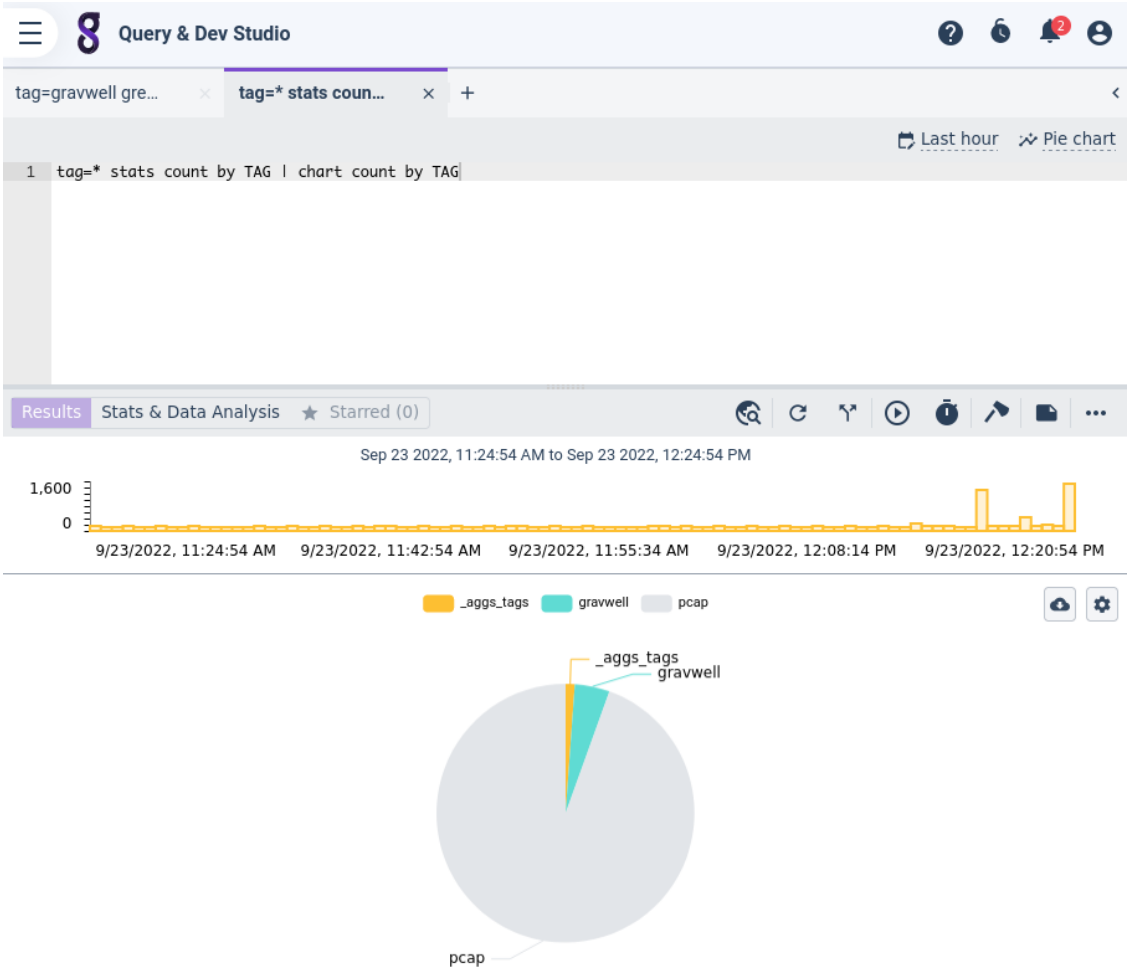


Figure 3.21: Query Studio results display

3.5 Playbooks

Playbooks are hypertext documents within Gravwell which help guide users through common tasks, describe functionality, and record information about data in the system. Most Gravwell kits (see Chapter 9) include a playbook or two to help users get oriented in the kit, but regular users can also *create* playbooks for themselves, documenting their data investigations with a mix of text, images, and *executable queries*.

The Playbooks page (Figure 3.22) lists the playbooks currently in the system and allows the creation of new ones.

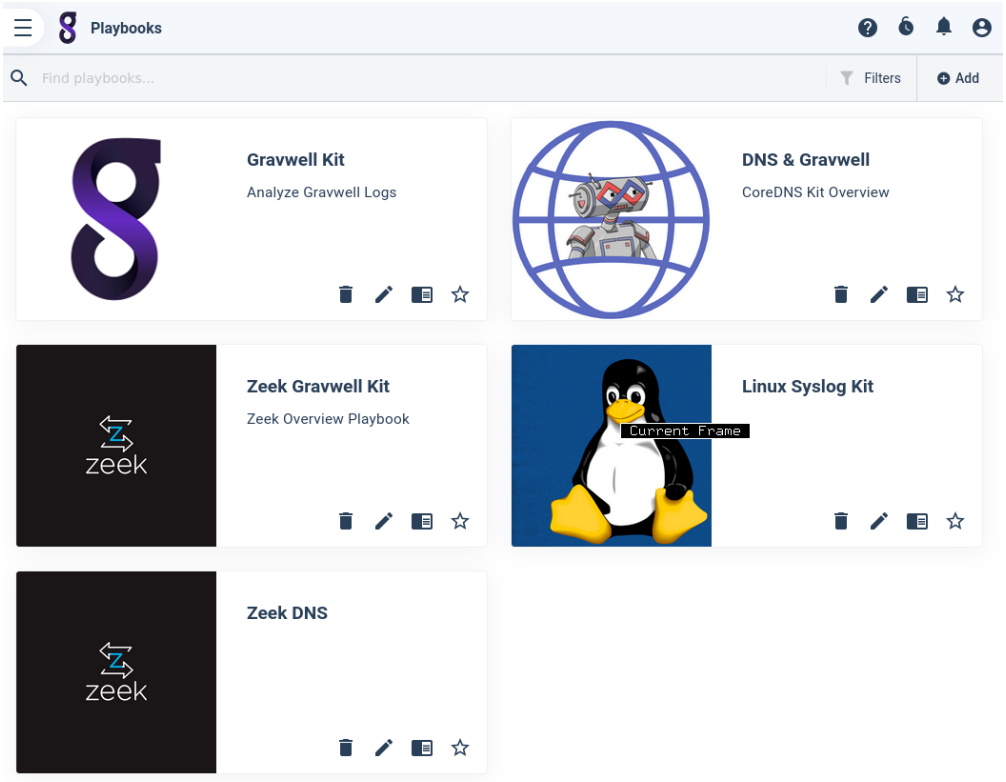


Figure 3.22: Playbooks page

Clicking a playbook will open it. It will look more or less like a regular web page, with section headings, hyperlinks, and images, but it will also include embedded queries, as seen in Figure 3.23. Clicking the 'Launch' button will run that query in a new browser tab.

3.5.1 Playbook Markdown

Playbooks are written in Markdown. You can edit an existing playbook by clicking the edit button, or create your own new playbook. This will bring up an editor as shown in Figure 3.24

The editor accepts standard Markdown syntax[1]:

- Designate headings with number signs (#), using one for heading level 1, two for heading level 2, and so on, e.g. `# Heading 1`, `### Heading 3`.
- Surround text with asterisks to italicize, or with doubled asterisks to bold, e.g. `*italic*`, `**bold**`.
- Hyperlinks put the link text in square brackets and the target URL in parentheses: `[Example Site](http://example.com)`

[1] https://www.markdownguide.org/basic-syntax/

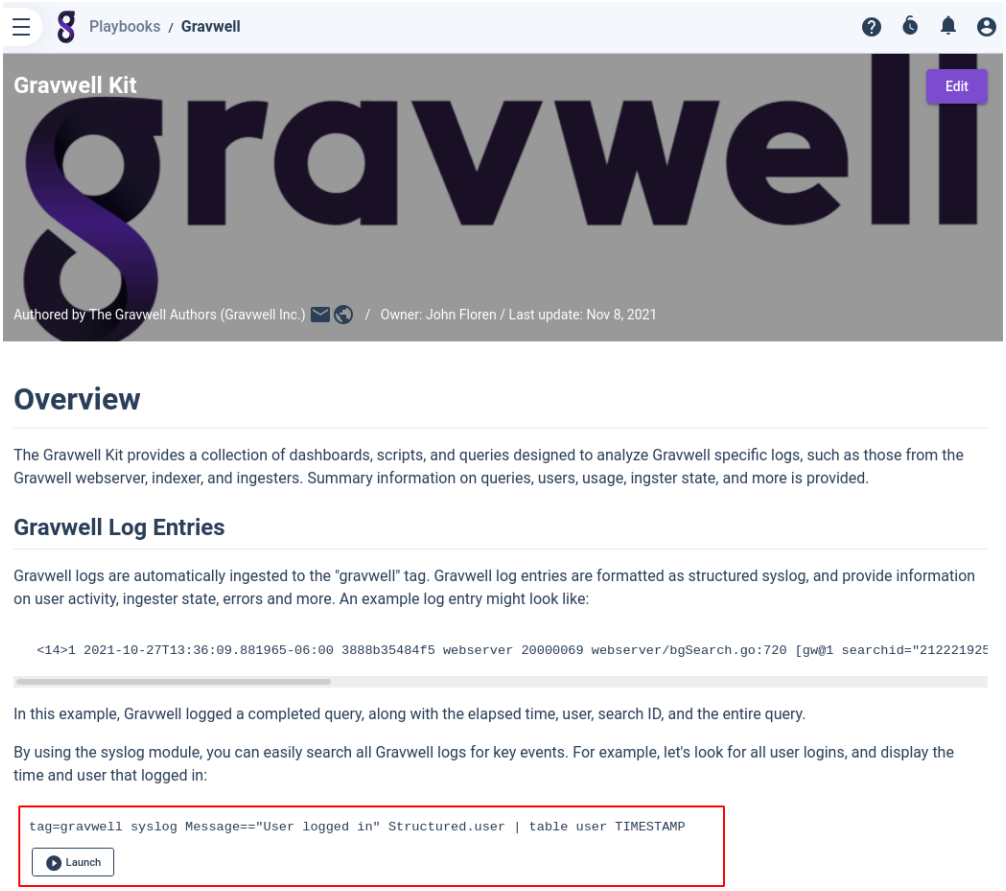


Figure 3.23: Playbook viewer

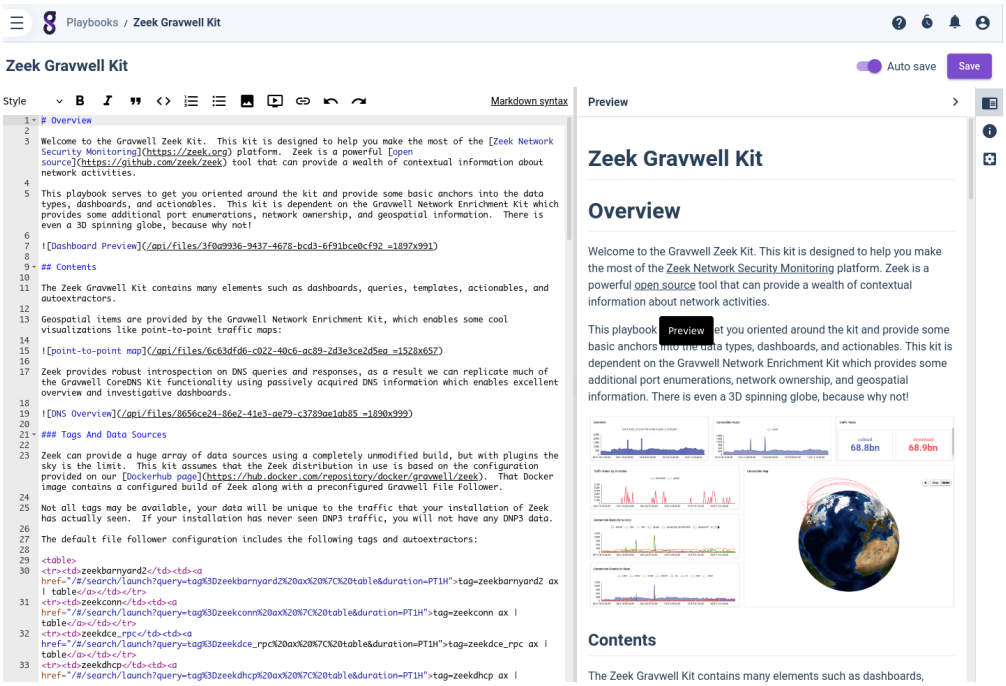


Figure 3.24: Playbook editor

- Create numbered lists by numbering each line (`1. First item` and so on), create bulleted lists with asterisks (`* First item`).

Review the full Markdown syntax document for more options. You can also insert plain old HTML if necessary. Note that there are two essential changes to the syntax which are explained below.

Inserting Runnable Queries

Playbooks can contain Gravwell queries which will be displayed with a "Launch" button for the user to click. To insert a query, use the Markdown blockquote syntax, as below:

```
```
tag=netflow netflow Protocol Src SrcPort Dst DstPort | table
```
```

Inserting Images

Images can be inserted in two ways. The first is the standard Markdown method using a hyperlink:

```
![This is the image alt text](https://example.com/image.jpg)
```

The other method involves uploading an image *to Gravwell*. To do this, click the 'Add image' icon in the playbook editor. The UI will prompt you to upload an image as seen in Figure 3.25. Once you select a file to upload, the dialog will prompt you for alt text and other information as shown in Figure 3.26. Once you click 'Add', the editor will insert Markdown code referring to your newly-uploaded file within Gravwell:

```
![A cat](/api/files/c39a9541-971f-44b1-97ac-5724d5214f05 =1240x698)
```

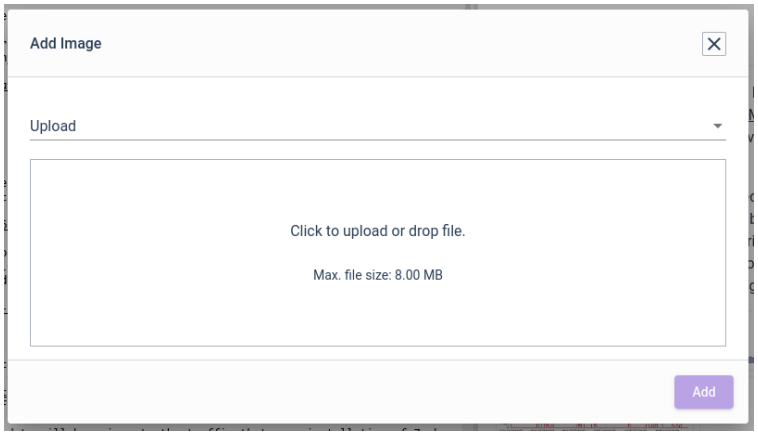


Figure 3.25: Playbook image upload

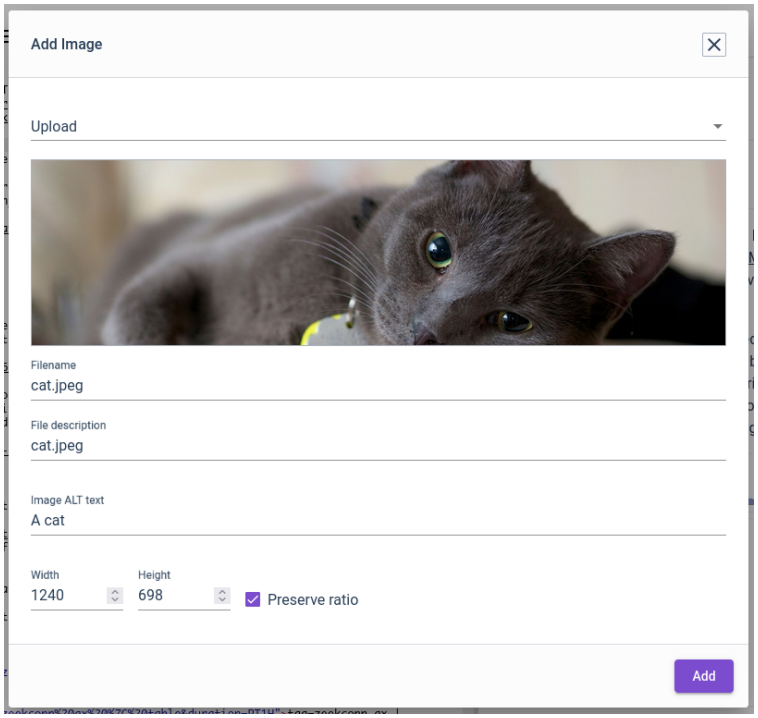


Figure 3.26: Playbook image upload

You can also insert a previously-uploaded image by selecting 'Gallery' in the dialog's drop-down menu, as shown in Figure 3.27.

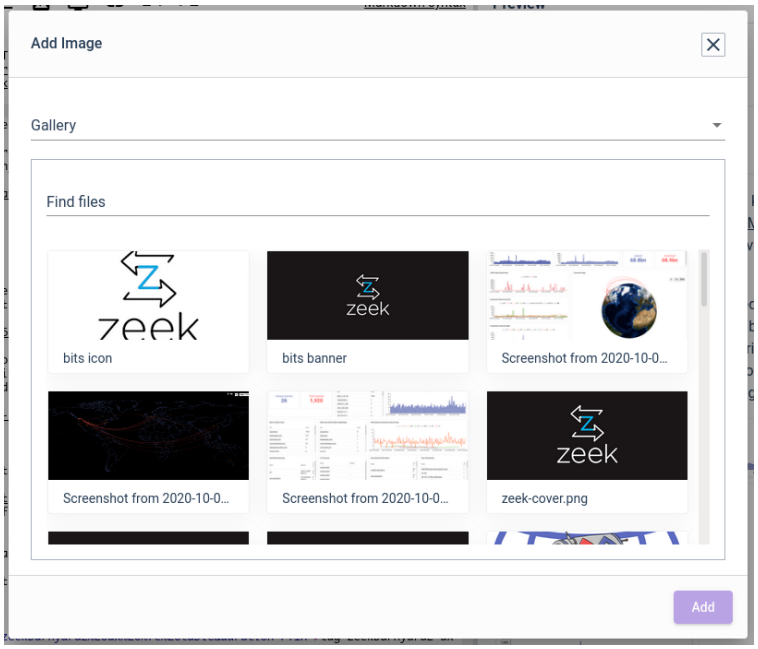


Figure 3.27: Playbook image gallery

Chapter 4

Searching

4.1 Search Pipeline Architectural Overview

Gravwell search is designed to be an asynchronous pipeline that behaves somewhat like a stream processor. The pipeline is transparently distributed across indexers, enabling distributed search without the user needing to explicitly define or direct specific search parameters. From a user perspective, a search operating on 1 node looks exactly the same as a search operating on 300 nodes. However, having a better understanding of how the pipeline operates can help craft queries that can better leverage the power of parallelism available in Gravwell. We have seen customers reduced query execution times by an order of magnitude with only small tweaks.

A pipeline is instantiated for every search. It intelligently identifies where data is located, decodes raw entries, extracts features, condenses, and finally readies the results for display. The pipeline is made up of the following elements:

- Tags. A tag is a data group which informs the indexers what set of data to feed into a pipeline. Indexers use tags to transparently bind to the appropriate wells in order to feed data to the pipeline.
- Data, composed of entries. These entries are stored on disk by indexers until a search is run.
- Search modules. These apply structure, extract elements, filter, and condense. Search modules do most of the "heavy lifting" in the pipeline.
- Render module. Any given pipeline has only one render module which must be the last module. Once entries have been processed by the search modules, the render module formats them to display to the user. If no render module is specified in a search, the text module is used.

Figure 4.1 shows an abstracted visualization of how a search pipeline functions. Raw data (marked in red) is stored as entries on the indexers. When a search begins, the indexers read the entries off the disk and feed them into a series of search modules (shown in blue). This allows the indexers to take on some of the computational load, in this case by executing the first three search modules on the indexers. However, eventually entries will hit a search module that requires *all* entries in order to perform its task; this is called a **condensing** module. Condensing modules perform tasks like counting, calculating means, and sorting; they condense because they require the complete data stream to do the operation. In order to condense, the entries must be shipped to the webserver where the condensing module is executing. In this example, once the webserver gathers entries from each of the indexers and joins them, it feeds the entries through three additional three search modules. Finally, the fully-processed entries are fed into the renderer which stores the results so that they can be presented to a human, script, or outside API.

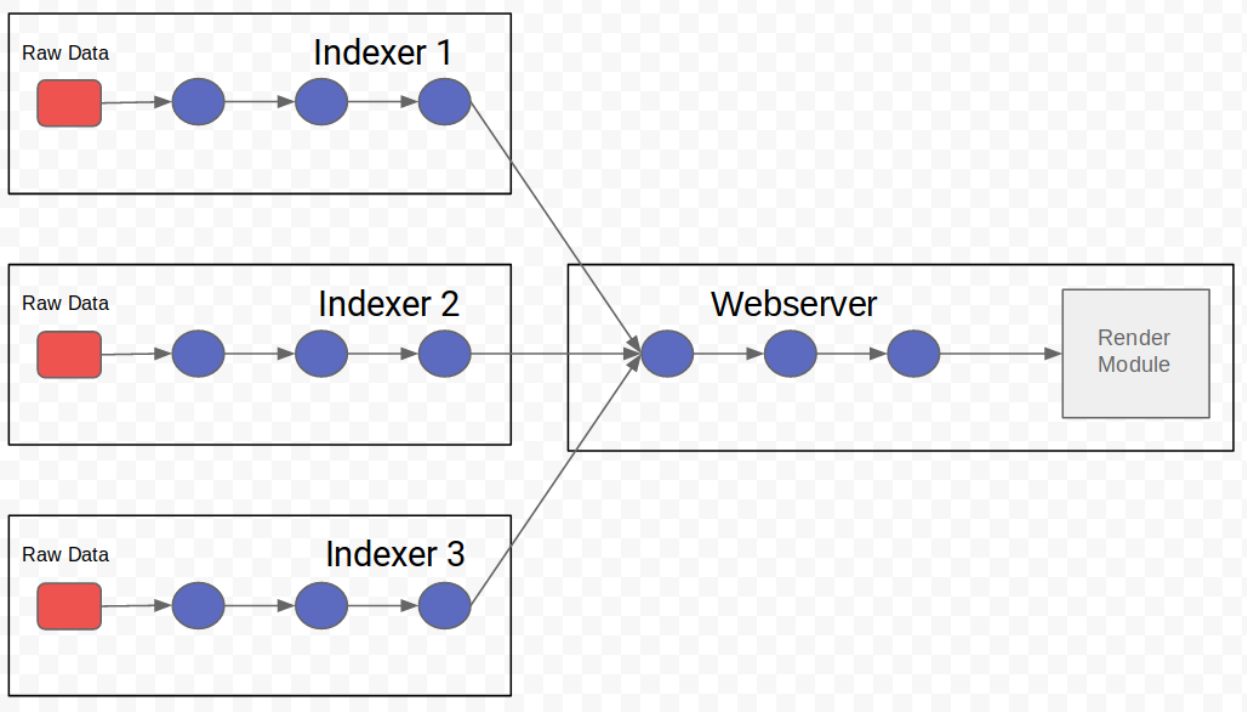


Figure 4.1: The Gravwell pipeline

4.2 Query Entries and Tags

The first part of any query is the *tag set*; all queries operate on at least one tag, but a single query can operate on multiple tags. If no tag is specified, the `default` tag is assumed. Every entry in Gravwell is assigned a tag when it is ingested. The tags control which well the entry is stored in and are kept with the entry throughout the entire query pipeline. To specify a set of tags for a specific query, prepend the query with `tag=` and a comma-separated list of tags. For example, if we wanted to examine all entries that have been assigned the tags "syslog" and "kernel" we would prepend our query with `tag=syslog,kernel`. Tags can also be specified using glob wildcards, so if our Gravwell deployment has data tagged "syslog", "system", and "netflow" we can select the tags "syslog" and "system" (excluding "netflow") by using `tag=sys*`.

The query `tag=*` instructs Gravwell to just go get all the entries, from every tag, and push them into the `text` renderer (the default renderer). A word of warning: if your system has lots of data and you run this query over a long period of time, you are basically asking all your indexers to dump their raw data to the webserver and hand it to your web browser. This will put a lot of traffic on the network and consume a lot of resources on the indexers and webserver.

We can expand on this query and use the `TAG` and `DATA` keywords to draw a table with the raw data and tag name. You can always access the human readable name of a tag through the TAG keyword.

```
tag=* table TAG DATA
```

Figure 4.2 shows an example of the output from this search. The table renderer will attempt to present data as a text, but since the data tagged as `pcap` is binary (actual packets) it can't really be printed as text, thus the "noise". Later in this chapter, we will show how to process this binary data.

Figure 4.2: Table showing tags and raw data

4.3 Chaining Multiple Modules

Gravwell search modules are each designed to accomplish a specific task; the power of Gravwell is obtained by chaining multiple modules together to accomplish something complex. Modules are chained together using the pipe character "|". You can chain as many modules together as you want. The Gravwell pipeline is designed to be asynchronous, with each module runs in its own lightweight thread. This means that if you build a query with 8 modules chained together, the query can potentially leverage 8 CPU cores for the work. This asynchronous execution allows Gravwell to leverage modern processors with many CPU cores. It also makes for faster execution even though each query module does not execute at the exact same speed. If you are familiar with a Unix command line and chaining multiple programs together to achieve a result, Gravwell will feel like second nature–the Gravwell pipeline is directly modeled on the Unix command line.

Let's examine a query that uses successive filtering modules to find syslog data from a specific service originating from a specific machine. We will use the `grep` module twice, first filtering for the machine name and then filtering for the service name. Gravwell's `grep` module behaves much like the Unix tool of the same name: it searches text for patterns the user specifies. See https://docs.gravwell.io/#!search/grep/grep.md for more information or refer to Section 4.3.1. Our example query is:

```
tag=syslog grep dunkel | grep sshd
```

This query tells all indexers that we only want to look at syslog data (`tag=syslog`). The first `grep` search module filters out all entries which do not have the value "dunkel" (a hostname) in their data. The second `grep` filters out all entries which do not have the value "sshd" in their data. The output entries are guaranteed to contain both "dunkel" and "sshd" in their bodies.

Therefore, given the following entries as input, the first one would be returned while the second would be dropped:

```
Mar 20 17:41:08 dunkel sshd[26779]: Failed password for root from 218.92.0.192 port 38978
↪  ssh2
Mar 20 17:41:05 porter sshd[21703]: Failed password for root from 218.92.0.192 port 57623
↪  ssh2
```

However, the query as you see it is not what actually gets executed. Our example query does not specify a *renderer* module, so Gravwell transparently appends the default text renderer. The search pipeline also detected that there were no condensing modules and the renderer did not report that it could condense, so it also injected "sort by time desc" so that all the entries are shown in the correct temporal order. The actual executed query is:

```
tag=syslog grep build | grep sshd | sort by time desc | text
```

You don't need to really worry about the injected modules and the default renderer, just know that if you don't specify a renderer Gravwell will show you the raw entries and ensure they are in the correct order.

4.3.1 The grep Module

Grep is a very basic pipeline module that searches for a text string (not Unicode). Any record containing such text will match and be passed through the pipeline. Any record not containing the text is dropped from the pipeline. For example, `grep foo` will pass on any records containing the text "foo" and drop any records that do not contain "foo" anywhere. Grep is case sensitive by default, so `grep foo` would match "foo" but drop "Foo".

Grep supports the standard GNU wildcards as well as fast string and binary matching. For instance to look for all entries that start that contain "foo" and "bar" separated by 0 or more bytes, you can use `grep foo*bar`.

Grep allows multiple patterns to be specified. If any pattern is matched, the entry is passed down the pipeline. If the -v flag is used to invert the search, the entry will be dropped if any pattern matches.

The module supports the following option flags:

- `-v`: "Inverse" grep. For instance, `grep -v bar` would drop any records containing the text "bar" and pass on any records that do not contain "bar".
- `-i`: Match case insensitive values. Thus, `grep -i foo` would match "Foo" and "foo". Case insensitive search tends to be one of the slowest operations; put it later in your pipeline if possible to keep things fast.
- `-e <arg>`: Operate on an enumerated value instead of on the entire record. For example, to show packets that contain HTTP text but aren't destined for port 80, run `tag=pcap packet ipv4.DstPort!=80 tcp.Payload | grep -e Payload "GET / HTTP/1.1"`
- `-s`: Strict match. All patterns must match, or in the case of a negated strict match, no pattern may match.‘
- `-simple`: Simple match. With this flag, grep will match exactly the characters you specify, with no wildcard matching. This allows you to find asterisks and other normally-reserved characters: `grep -s *`
- `-w`: Word match. The entire match pattern must be a full word as would be matched by the fulltext extractors.

4.3.2 Hands-on Lab: Basic Filtering

For this hands-on lab we are going to chain a few modules together to achieve some basic filtering and zero in on very specific data.

Start by cleaning your environment and starting up a new base Gravwell instance:

```
docker stop $(docker ps -q)
docker rm $(docker ps -q -a)
docker create --rm --net gravnet -p 8080:80 --name test gravwell:base
docker start test
```

Next, we'll use the `reimport` ingester to ingest a prepared set of logs into the `syslog` tag:

```
cd ~/gravwell_training/Search/Lab-Basics
docker run -v $PWD/data:/tmp/data --rm -i --net gravnet \
gravwell:ingesters /opt/gravwell/bin/reimport -rebase-timestamp \
-clear-conns test -i /tmp/data/auth.json.gz -import-format json
```

Log into your Gravwell GUI (http://localhost:8080, recall that the username/password is "admin"/"changeme" by default), then try the following tasks:

1. Using the `grep` module, filter the syslog tag to only include logs from the sshd daemon on the host `porter`.
   (a) Hint: Refer back to the section above for example queries you can adapt.
2. Filter the syslog tag to include sshd logs from all hosts *except* `porter`.
   (a) Hint: The grep module has a -v option that inverts the filter logic, meaning any entries which match the pattern will be *dropped*.
3. Filter the syslog tag to find successful logins on the host `porter`.
   (a) Hint: sshd will log "Accepted" when a user logs in.
4. Generate a final count of failed ssh password logins for non-root accounts.
   (a) Hint: Use the `count` module without any arguments to count entries.

Do not `rm` the test container, as we are going to use it again for the next lab.

4.4 Entries, Enumerated Values, and Field Extraction

One of the most common operations any Gravwell user will perform is field extraction. Gravwell is designed to operate on unstructured data, meaning that you don't necessarily have to understand the form of your data until you need to actually query it. Field extraction allows us to isolate and extract the tidbits that we care about and then operate on them to gain useful insights.

When a field is extracted from an entry, it becomes an **Enumerated Value**. Enumerated values are composed of a name, data, and a type. In most cases, you don't need to worry about the type since Gravwell will handle the translations and casting for you, but should you run into any ambiguity in a query the type is there to help Gravwell do the right thing. When an enumerated value is extracted it is attached to an entry for the remainder of the pipeline. Entries can have many enumerated values attached.

The best way to demonstrate field extraction and enumerated values is to perform a few queries and look at the results. Let's look at the same syslog data we used previously. Here's an example:

```
Mar 20 23:59:48 porter sshd[35522]: Failed password for root from 218.92.0.151 port 26814
↪  ssh2
```

A syslog entry consists of a date, the hostname of the system that sent the message, the application that generated the message, the process ID of that particular application, and the message itself. Sometimes a message may contain additional information, but these basic fields should always exist.

We can use Gravwell's `syslog` module to extract some of these fields as enumerated values and display them in a table. Figure 4.3 shows the results.

```
tag=syslog syslog Appname Message | table
```

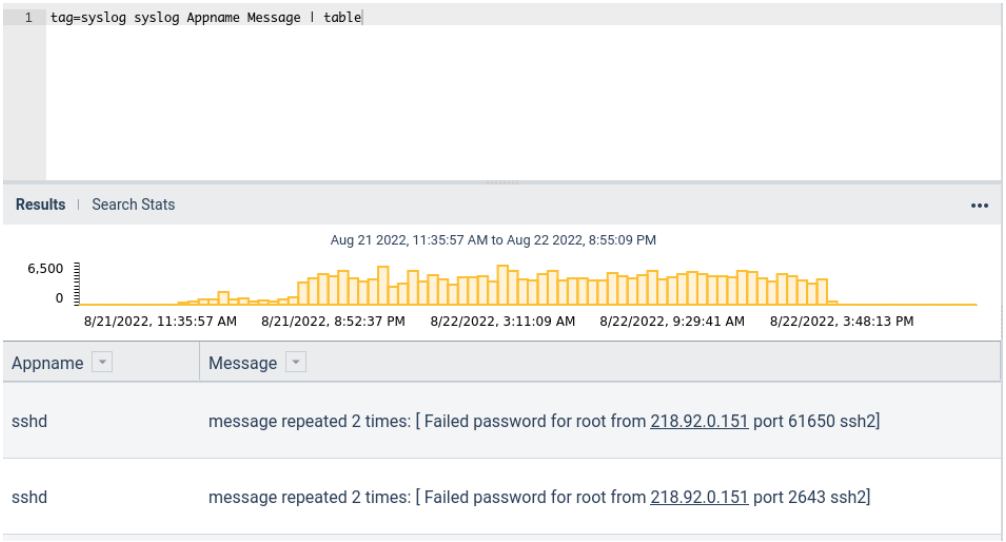


Figure 4.3: Fields extracted from syslog entries.

Note that if you specify the table renderer and don't give it column names, it will automatically use every enumerated value created in the pipeline. The syslog module extracted features from the log entry named "Appname" and "Message" and attached them to the entry. The table renderer then created columns using the enumerated value names and rendered a row for each entry, populating the columns with the enumerated value contents.

We can expand on these extractions by also extracting the method and URL. Rather than modifying the original `syslog` command, we can chain in another one to extract the additional values. This demonstrates that successive modules can extract additional features and attach them to entries.

```
tag=syslog syslog Appname Message | syslog ProcID Message | table
```

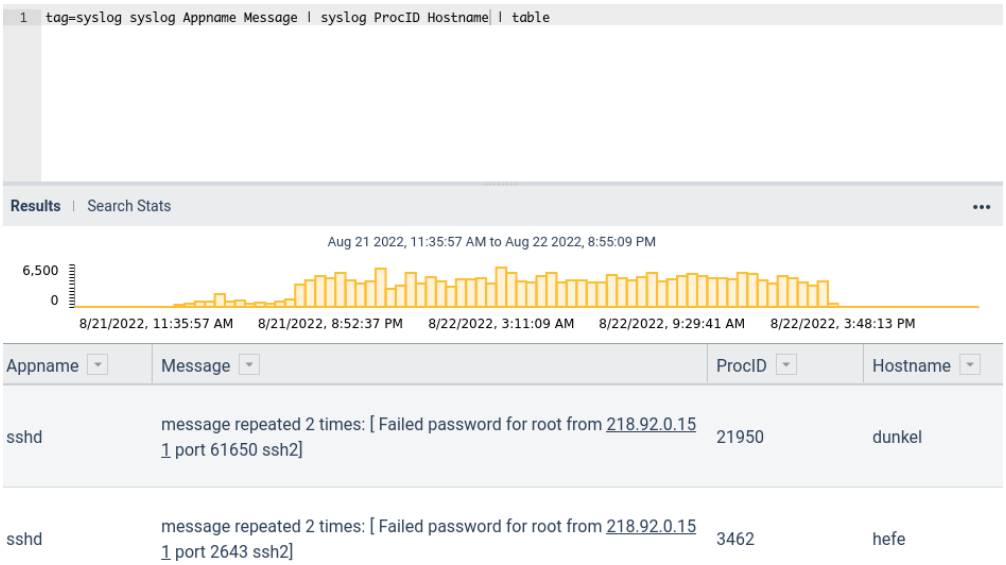


Figure 4.4: Chaining multiple extractions

Now that we have the Appname, Hostname, ProcID, and Message, let's extract a feature from a feature. This data contains lots of messages about failed login attempts in the Message fields; we can use the `regex` module to extract the username and IP address from these failed login attempts with a regular expression. We specify the `-e` flag to tell `regex` that it should look *only* at the contents of the Message enumerated value, rather than the entire raw entry.

We'll simplify the query a little by condensing the two `syslog` invocations into one, and we'll also tell the table module to only display the columns we're interested in. Figure 4.5 shows the results of the query.

```
tag=syslog syslog Appname Message ProcID Hostname
| regex -e Message "Failed password for (?P<username>\S+) from (?P<ip>\S+) port"
| table Hostname username ip
```

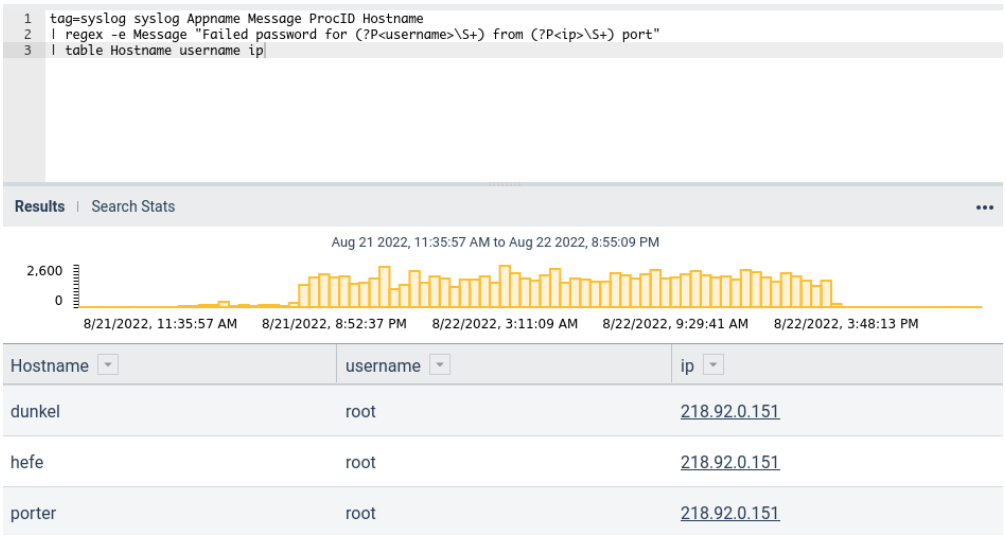


Figure 4.5: Extracting from an enumerated value

We were able to chain multiple pipeline modules to extract multiple fields and then operate on those extracted fields to make a nicely-formatted table showing failed login attempts. We can even invoke the `count` module

to track how many attempts occurred for each user on each system, as seen in Figure 4.6. Note how we specify `by Hostname username`, telling the `count` module that it should keep a separate count for each unique combination of Hostname and username.

```
tag=syslog syslog Appname Message ProcID Hostname
| regex -e Message "Failed password for (?P<username>\S+) from (?P<ip>\S+) port"
| count by Hostname username
| table Hostname username count
```

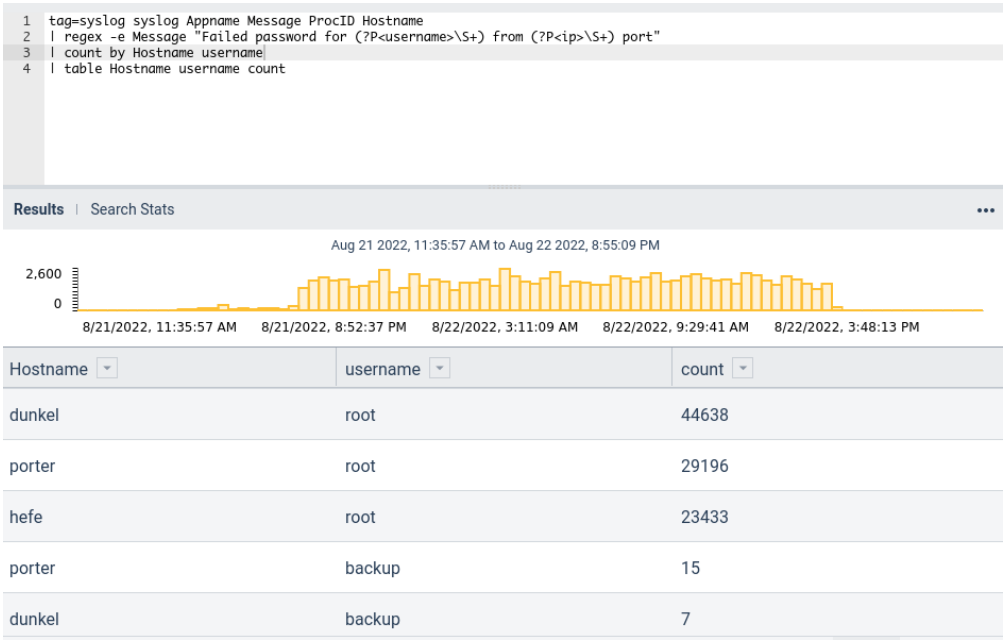


Figure 4.6: Doing statistics with enumerated values.

4.4.1 Hands-on Lab: Searching with Enumerated Values and Field Extraction

For this lab we are going to be using the same data set as the previous lab (Section 4.3.2), but instead of just filtering we will extract data and output statistics to zero in on some bad behavior. If you tore down the environment from previous lab, go back and stand it back up and reingest the auth data.

We will be using the syslog module to extract data from Linux auth logs as demonstrated in the preceding section. You'll use the `syslog`, `grep`, and `count` modules, along with the `table` renderer.

Lab Tasks

1. Extract the `Appname` and `Hostname` fields from the entries and put them in a table.
   (a) Add another extraction to pull the `Message` portion of the logs and add it to the table.
2. Filter to only include sshd messages by matching on the `Appname` enumerated value. Hint: `grep` takes a `-e` flag to specify an enumerated value!
3. Generate a table with the count of `Appname` events by each `Hostname`. Hint: `count by host`
4. Extract the `ip` and `username` for all SSH login events. Hint: refer back to the regex examples given in this section.

BONUS: generate a table with success and failure counts for each IP.

To clean up after the experiment, simply run:

```
docker kill $(docker ps -a -q)
```

4.5 Inline Filtering

Many search modules support what we call "inline filtering". Inline filtering allows the module to extract a value and filter on it using its native type without involving another module. There are some real benefits to using inline filtering as opposed to extracting in one module and filtering in another, the most important which may be acceleration. Search modules that support inline filtering know how to communicate the filters to an indexer's acceleration engine, which can enable dramatic speedups. Even when not filtering in a manner that can invoke the acceleration engine, inline filtering allows for fast type-native operations.

Let's start by examining some modules that support inline filtering and examine a query that would not invoke the accelerators and then adapt it so that it can invoke the accelerator.

Let's begin by examining the data:

```
tag=json
```

Here is an example of the data we're looking at:

```
{
      "time":"2022-06-08T19:14:31Z",
      "account":{
         "user":"miawilson242",
         "name":"Michael Moore",
         "email":"miawilson242@test.org",
         "phone":"+81 43 3387494435",
         "address":"15 Lincoln Rd,Newstead, OH, 13188",
         "state":"IA",
         "country":"Brazil"},
      "class":40234,
      "groups":["dagger","jester"],
      "user_agent":"Mozilla/5.0 (Windows NT 6.3; Trident/7.0; rv:11.0) like Gecko",
      "ip":"194.50.167.70",
      "data":"But what sort of language would we have the world speak, if we were told
      ↪  the miracle of Babel was presently to be reversed?"
}
```

The data generated is random. Let's take a look at the country and groups fields in the example above:

- country

  "Brazil"

- groups

  "dagger"

  "jester"

Let's say we were looking to perform some analytics and we wanted to see what the "groups" field looked like for users in Brazil.

Because groups is an array of values, we will extract the array and pass the output name to the `-x` flag:

```
json -x groups groups account.country
```

This will turn the single entry into two entries, one with enumerated values `country=Brazil` and `groups=dagger`, one with `country=Brazil` and `groups=jester`.

We might write the following query:

```
tag=json json -x groups groups account.country | grep -e country Brazil |
count by groups | chart count by groups
```

This query will show us a plot of activity groups that are registered in Brazil. But what if we have hundreds of thousands of users making making thousands of requests per day, literally hundreds of millions or billions of JSON records generated and ingested by Gravwell? Gravwell will handle those numbers, but the above query will not invoke the accelerators and as a result hundreds of millions of log entries we don't care about will be read off the disks. We can make a small adaptation to the query that will:

1. Let the `json` module inform the indexer about a filter (invokes acceleration).
2. Let the `json` module filter directly without extracting and passing every value.

```
tag=json json -x groups groups account.country==Brazil | count by groups |
chart count by groups
```

The new query doesn't look much different and the output is identical, but the performance may be dramatically different. It's also easier to read. If we have indexed on the `country` field, the json module can tell the acceleration system about our filter. As a result, only entries that have a country value of "Brazil" are ever retrieved from the disk or enter into the pipeline.

Table 4.1 shows the filtering operations supported by Gravwell. Note that the greater than/less than operations can only operate on numeric enumerated values; Table 4.2 shows which enumerated value types are compatible with which filters. In this context, "subset" means that the enumerated value *contains* the argument, so `foo~"well"` would match the strings "well", "wellbeing", and "Gravwell".

Operator	Name
==	Equal
!=	Not equal
<	Less than
>	Greater than
<=	Less than or equal
>=	Greater than or equal
~	Subset
!~	Not subset

Table 4.1: Filter operations

Enumerated Value Type	Compatible Filters
string	==, !=, ~, !~
byte slice	==, !=, ~, !~
MAC address	==, !=
IP address	==, !=, ~, !~
integer	==, !=, <, >, <=, >=
floating point	==, !=, <, >, <=, >=
boolean	==, !=
duration	==, !=, <, >, <=, >=

Table 4.2: Filter compatibility

The subset and not-subset operators have a special meaning when the enumerated value is an IP address: they check if the IP address is part of a *subnet*. Thus `ip~192.168.0.0/16` evaluates to true *if* the value in `ip` is in the 192.168.0.0/16 subnet.

If we haven't indexed on the `country` field, we still get a speedup because the filter allows the json module to be more efficient. The `json` module has to walk the data and find each field it is asked to extract, but by specifying a filter directly in the json module it can stop execution if one of the extractions fails the filter. In the above query example, let's say it extracts the country field and gets "Mexico". With inline filtering, the

json module knows that this fails the filter, so it stops processing the entry; the `group` extraction is never even performed.

Let's look at another query where inline filtering lets us perform a filter that is more complex than simple equality, where the processing module can leverage its knowledge about the data to do something more interesting. Netflow is a network monitoring data format that generates flows (records of network connections) from raw network traffic. It will passively watch a network and report on connections, saying that IP A sent X bytes to IP B using the port Z. So instead of capturing potentially trillions of bytes for network statistical monitoring, we can capture just a few. Many smart switches and routers will natively export netflow. Using the netflow[1] search module, we can parse the binary Netflow format and do some analysis:

```
tag=netflow netflow Src~192.168.0.0/16 DstPort < 1024 |
count by Src DstPort | table Src DstPort count
```

For this query we are leveraging the fact that the `netflow` module knows it is dealing with an IP and port; this allows us to apply a filter with the context of those types. This query is applying a filter that says we only want flow records where the source IP address is in the Class B subnet 192.168.0.0 and where the destination port is a service port (below 1024). These inline filters are not equality filters, so they will not invoke the accelerators.

Inline filters also have one other trick: some of them can examine two different fields and extract the one that matches a filter. Assume we are monitoring Netflow on an internal network and we want to see all flows that use SSH. We don't care if the flows are inbound or outbound, but in order to filter for SSH we need to filter on either the SrcPort or the DstPort. The netflow module lets us specify that a filter match either the SrcPort *or* the DstPort by just specifying "Port". We can do the same thing for the Src and Dst IP addresses by just specifying "IP". So, if we wanted to adapt the above query to show us any internal private IP that is using SSH, it would look like this:

```
tag=netflow netflow IP~PRIVATE Port==22 Bytes |
sum Bytes by IP | chart sum by IP
```

This query also uses the PRIVATE keyword, which tells netflow we are looking for any non-routable IP[2] addresses. Also, even though the Port field might end up referring to either SrcPort or DstPort, because we are using an equality filter it can still invoke the accelerator (if we are accelerating on the field). If we were on a large network with many billions of flows per day, the accelerators would dramatically speed up that query, plus we get a cool chart to show us who is pushing a bunch of data around using SSH.

[1] https://docs.gravwell.io/#!search/netflow/netflow.md
[2] https://en.wikipedia.org/wiki/Private_network#Private_IPv4_addresses

4.5.1 Hands-on Lab: Inline Filtering

In this lab we will observe how the inline filtering system can improve query performance and simplify the query, especially when combined with acceleration (see Section 5.5 for more information on acceleration).

First, we'll get into the lab subdirectory:

```
cd gravwell_training/Search/Lab-Filtering/
```

Create a new container using the gravwell:base image, upload our lab-specific configuration file, and restart the container.

```
docker create --rm --net gravnet -p 8080:80 --name test gravwell:base
docker cp config/gravwell.conf test:/opt/gravwell/etc/gravwell.conf
docker restart test
```

Now we'll use the generators image to generate some JSON data; if you don't have the `gravwell:generators` image, see Section 2.5 for instructions on how to load it.

```
docker run --net gravnet --rm -it gravwell:generators \
/jsonGenerator -clear-conns test:4023 -entry-count 500000
```

Open your Gravwell GUI (http://localhost:8080) and check that there is a new well named "json" with 500k entries in it.

Let's start with a query which chains several modules together to filter in order to get a chart of user activity for all users that are registered in the country Mexico. We will use the json module to extract the country and email members, then filter it using the grep module to only look at users that are registered in Mexico and are using an email address from `example.com`.

Execute the following query:

```
tag=json json account.country account.email | grep -e country Mexico |
grep -e email "@example.com" | count | chart count
```

This query requires that the json module process every single entry, extract the country and email fields, and pass it to the grep module for filtering. We can verify that the pipeline processed 500,000 entries by clicking on the stats tab of the query results.

Lab Tasks

1. Adapt this query to use the `json` module's inline filtering.
   (a) Eliminate both greps.
   (b) Compare the time required for the query to using grep.
2. Create a new query that generates a list of all users in the group "myth" and from the country Canada.
3. Adapt the query to generate a count of unique users in the group "myth" for each country sorted by largest count first.

Note: If your queries do not yield results, review the data or your list of pre-selected values. Adapt the query to search for a different group or country; it is possible that the generated data may not include a particular combination of country and group.

To clean up after the experiment, simply run:

```
docker kill $(docker ps -a -q)
```

4.6 Data Exploration

Gravwell provides several tools to help make sense of data with little or no knowledge of the data itself or of the Gravwell query language. Broadly, we call these capabilities *Data Exploration*.

4.6.1 Word Filtering

The most basic form of data exploration is word filtering, in which the user clicks a word within the raw entry and either includes it or excludes it from the search. For example, if we run the query `tag=gravwell`, we'll see a bunch of textual results. If we move the mouse cursor over the results, individual words are highlighted. We can click on a highlighted word to bring up a context menu which allows us to *include* or *exclude* the selected word from the search, as seen in Figure 4.7.

Figure 4.7: Highlighting words in text results

Note: In Figure 4.7, the string "search.go" is also highlighted by an actionable (explained in Section 4.17); the menu item "search.go" contains a sub-menu with the actionable's actions in it.

If we click "Include search.go:189", the query will be automatically modified to 'tag=gravwell words search.go:189'. We can then re-run the query to see the new results. We can do this multiple times, including and excluding words, and the query will continue to update (Figure 4.8).

Note that if we have written a query already, we can still click on words and have them properly inserted into the query. For instance, if we manually run the query `tag=gravwell syslog Appname!=webserver` and then click on the word "flow" in the results, the query will be automatically rewritten to `tag=gravwell words flow | syslog Appname!=webserver`.

4.6.2 Field Extraction

The Query Studio (see Section 3.4) also has the ability to parse many data formats and split out individual fields. For example, a user may run the search `tag=gravwell` just to see what the raw entries look like. Clicking the Details Pane icon (Figure 4.9) tells Query Studio to attempt to parse the data and split out individual fields.

If this is the first time the user is looking at this tag, Gravwell must determine the most appropriate format for parsing the data. A window will appear presenting several options (Figure 4.10).

In this case, Gravwell believes the data to be syslog-formatted and shows a preview of the first result parsed as syslog. The user examines this result and, deeming it acceptable, clicks "Select" and saves the selected extraction.

Now the UI shows the Details Pane at the bottom of the window, with individual fields broken out for easier reading (Figure 4.11). Note the purple bar on the left side of the raw entry; that indicates which entry is currently being displayed in the Details Pane.

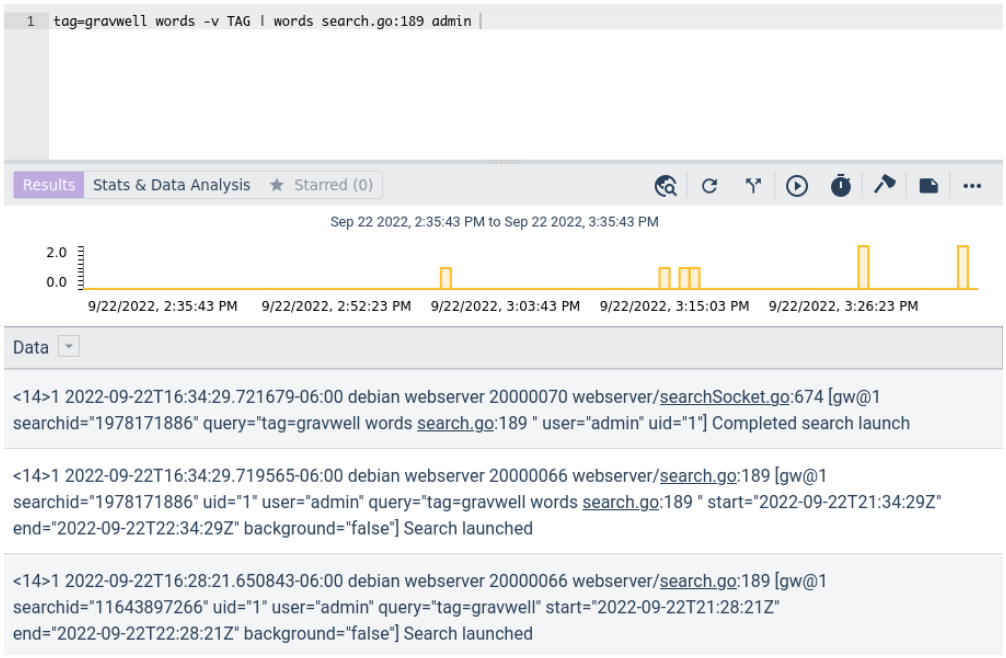


Figure 4.8: Results filtered by selecting a word

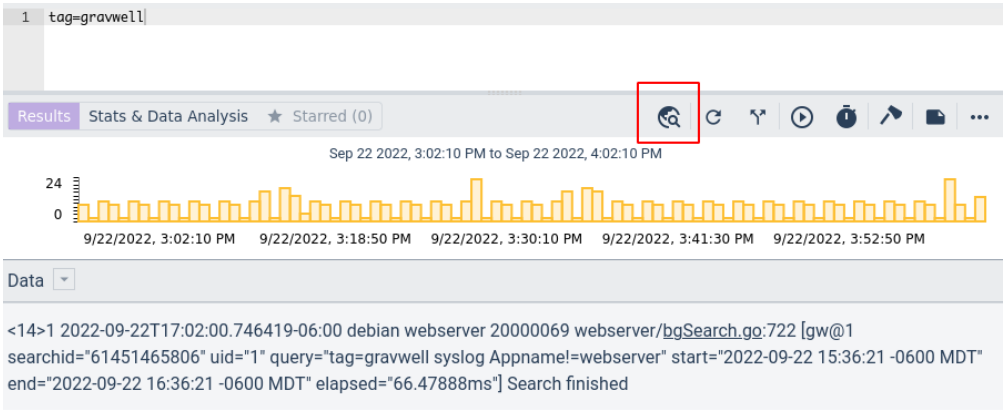


Figure 4.9: Selecting the details pane in Query Studio.

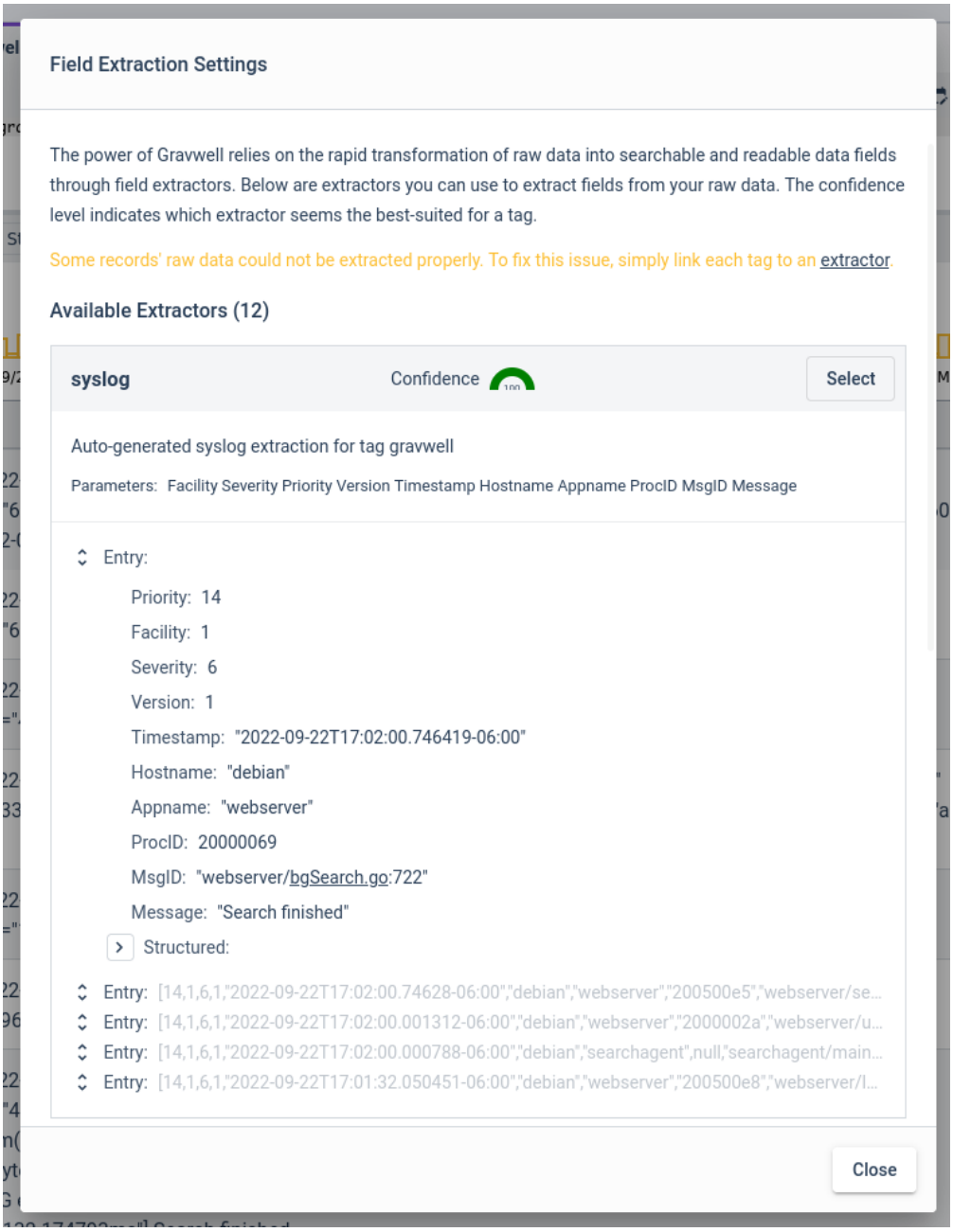


Figure 4.10: Selection of possible data extraction format.

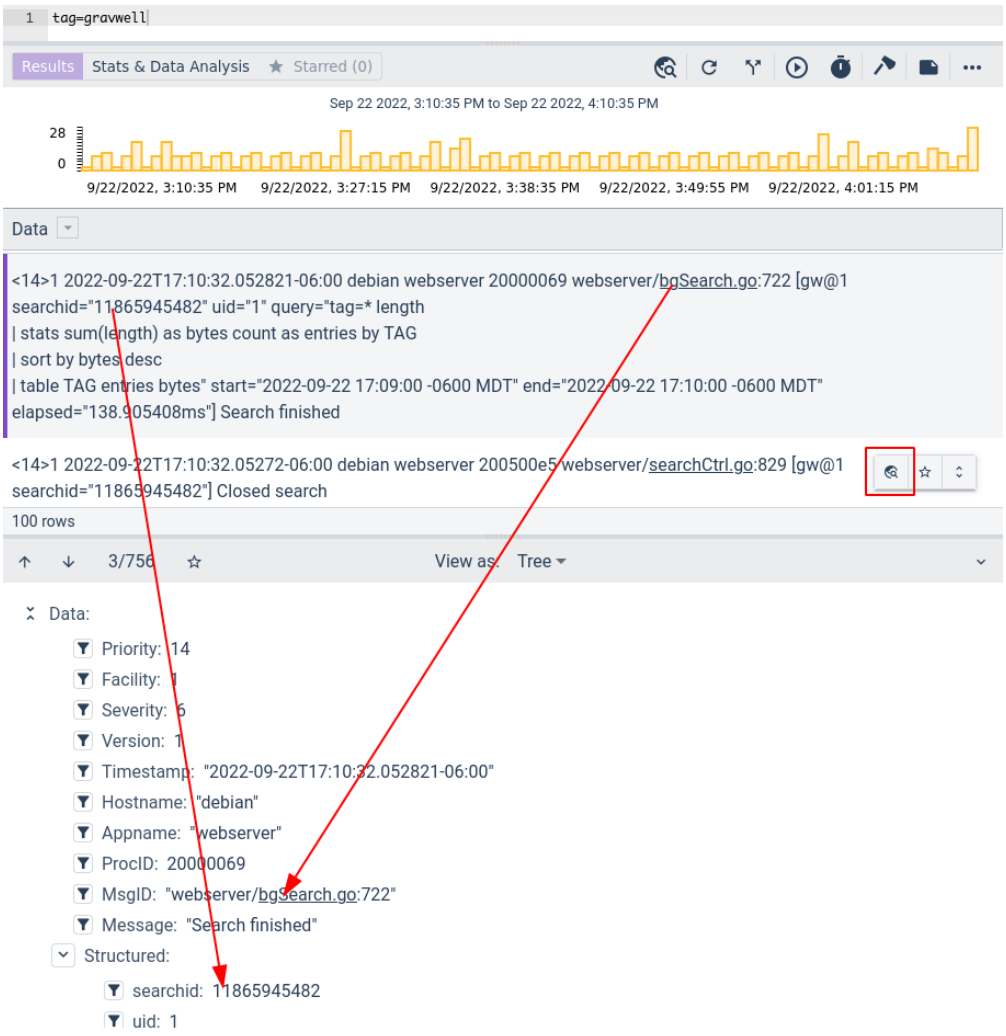


Figure 4.11: Viewing details for a single entry.

Each entry, when pointed to with the mouse cursor, displays the Details Pane icon in a floating button group. Clicking that icon will switch the Details Pane over to show the fields of *that* entry.

Within the extracted fields of the Details Pane, you can click the filter icon and add filters on the given field. In Figure 4.12, we are adding a filter to *exclude* any messages where the Appname is "webserver".

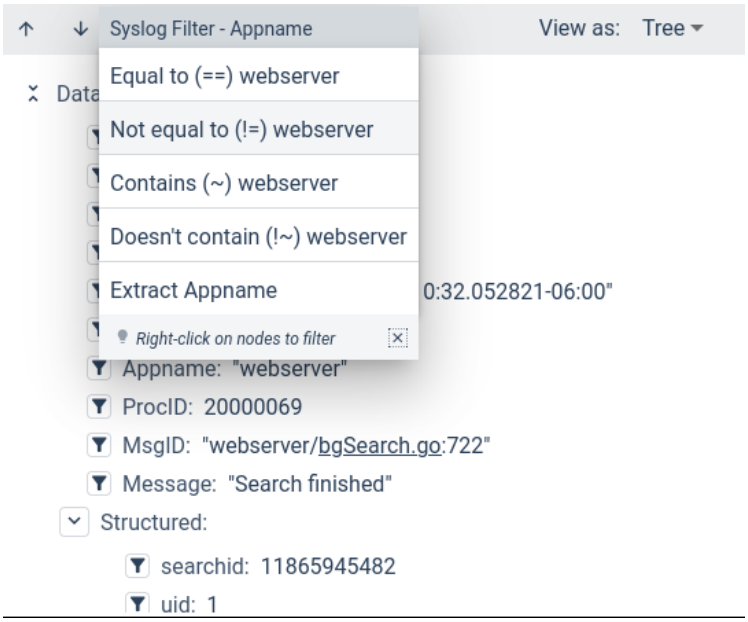


Figure 4.12: Adding a filter.

Selecting the menu option will automatically update the query with a filter on the specified field. Re-run the query to get the updated results (Figure 4.13). You can then continue in this way, selecting fields from the Details Pane (or clicking on words in the raw entries, see subsection 4.6.1!) to drill down into precisely the data you care about.

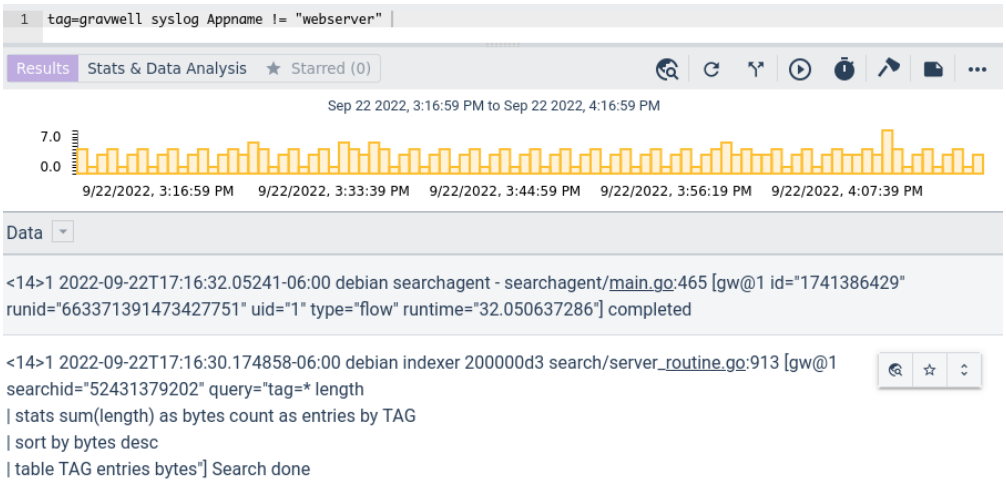


Figure 4.13: Query with filter inserted.

4.7 Search Modules

Gravwell supports a wide variety of search modules that are designed to perform various functions. Many of the search modules are used to perform feature extraction such as `json`, `syslog`, `cef`, `csv`, `fields`, `netflow`, etc. Feature **extraction modules** transform raw data into structured data so that features can be extracted; they are also designed to be specification tolerant. This means that the modules will often operate on data that doesn't quite fit the spec. The `cef` and `syslog` modules are great examples, as there are specs for syslog (RFC5424) and CEF but almost no one follows them completely, so the extraction modules have to be tolerant to what is actually out in the wild.

Gravwell also has many **analytical modules**. These modules perform statistical analysis. Examples of these are `count`, `sum`, `mean`, `stddev`, `variance`, and `stats`. The analytical modules are great for performing analysis to identify trends, deviences and abnormalities in very large datasets. Many problems have been identified simply by counting log entries over time. Other search modules allow for plugging in logic which can allow for very complicated processing. The `eval` module allows for arbitrarily complex boolean logic, and the `anko` module lets you plug a Turing-complete script into the pipeline. If there isn't a module that accomplishes exactly what you want to do, Gravwell makes it possible to drop in your own custom logic via these modules.

Search modules can also perform **feature enrichment**. Using the `lookup` module we can add external context to data as it goes by (such as assigning a hostname to an IP). The `langfind` module can use statistical analysis to infer the language of text. Using Gravwell you might be able to classify users by spoken language using chat logs, messages, or emails. Gravwell also has modules for complex machine learning tasks; we used Bayesian inference to generate datasets which classified social media posts as positive or negative (turns out people are pretty angry on the Internet). The Gravwell pipeline is designed to be pluggable and the available search modules are always expanding and improving. If you can't find what you want and aren't ready to roll your own using anko, Gravwell may be able to implement the module for you.

4.7.1 Extraction Modules

Extraction modules apply structure to raw entries to extract features. For instance, the `syslog` module will look at an entry and attempt to parse it as a Syslog message. If successful, it can extract syslog features such as the hostname, application name, priority, message, and so on. The `netflow` module can look at binary data, parse it as Netflow records, and give you back IP addresses, port numbers, etc.

One important thing to note is that you can attempt to apply any extraction module to *any* data. You can use the `csv` module on Syslog messages, and while that does not *usually* produce much of value, it can for instance be a convenient way to parse Palo Alto logs that are wrapped in Syslog: the first "column" ends up including the entire Syslog header and the first field of the Palo Alto logs, which is *always* marked as 'reserved for future use'. Figure 4.14 shows how the CSV module can extract fields from a Palo Alto syslog entry.

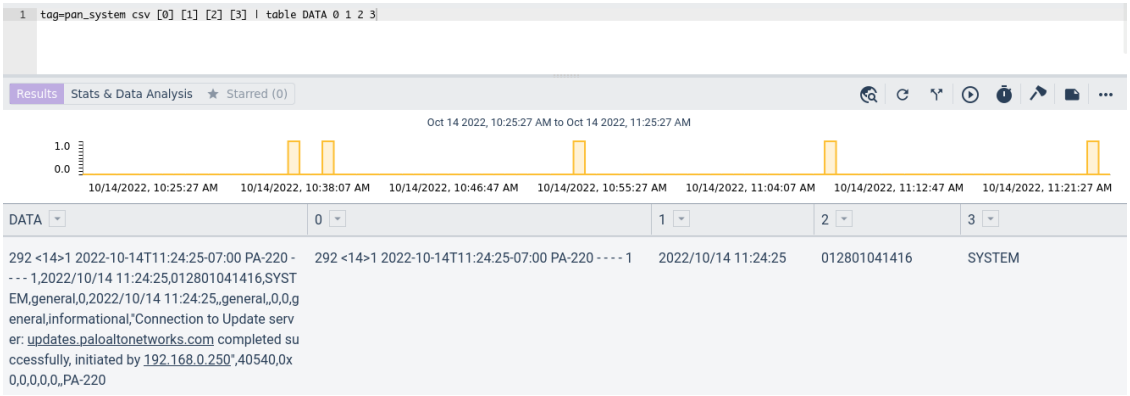


Figure 4.14: Parsing Syslog messages with the `csv` module.

Extraction Names and Paths

When an extraction module extracts a feature, it stores the contents of that feature in an enumerated value. If you say `tag=syslog syslog Appname Hostname`, the syslog module will extract the application name and hostname field from each entry and attach them as enumerated values named Appname and Hostname, respectively.

Some modules, like `netflow`, have a pre-defined set of which fields can be extracted: no Netflow record will ever contain any *other* fields. Thus, `netflow Src Dst` will work on *all* Netflow data, and `netflow foo` will *never* work. These pre-defined fields are documented in the Gravwell documentation wiki, and will usually be "hinted" when you are typing a query into the user interface.

Other modules, like `json`, deal with self-describing data and can therefore extract arbitrary fields. If an entry contains the following JSON:

```
{ "foo": { "bar": 3 } }
```

Then the invocation `json foo.bar` will extract an enumerated value named `bar` containing the number 3.

Other modules, like `csv`, can extract arbitrary fields, but the data is *not* self-describing. Consider the following CSV entry:

```
webserver,157,shutdown
```

We can extract the individual fields with `csv [0] [1] [2]`, resulting in the enumerated values shown in Table 4.3.

E.V. Name	Value
0	webserver
1	157
2	shutdown

Table 4.3: Enumerated values from `csv` module.

Gravwell doesn't know what those columns represent. However, the user can specify names for the columns using the *as* keyword, e.g. `csv [0] as component [1] as pid [2] as status`. Table 4.4 shows the result.

E.V. Name	Value
component	webserver
pid	157
status	shutdown

Table 4.4: Enumerated values from `csv` module.

Because it is useful to select your own names for fields, it is typically allowed in *all* extraction modules:

```
json foo.bar.baz as MyValue
netflow Protocol Bytes as traffic
```

4.7.2 Statistics Modules

The statistics modules in Gravwell deserve special mention. They are used in perhaps a majority of queries: one of the most common questions asked in Gravwell is "how many times did X occur?". The following statistical operations are available:

- count: count entries
- sum: sum values and return the total
- total: sum values over a period of time and keep a running total.

- mean: calculate the average
- stddev: calculate the standard deviation
- variance: calculate the variance
- min: return the minimum value seen
- max: return the maximum value seen
- unique_count: return the number of unique instances of the source were seen.

The simplest possible invocation of a stats operation is the following query, which counts the number of entries in the `gravwell` tag:

```
tag=gravwell stats count
```

You can do two operations at the same time; in the following example, we calculate each entry's length, then find the mean entry length and the total number of entries and display them in a table. Figure 4.15 shows the results of running this query, indicating that out of 12,057 entries, the average entry length was 209.73 bytes.

```
tag=gravwell length | stats mean(length) count | table
```

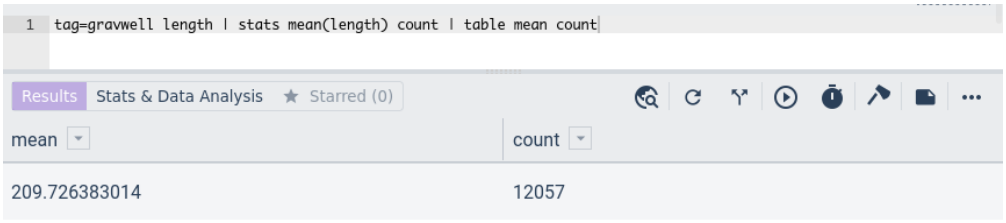


Figure 4.15: Multiple stats operations at once.

Note the syntax `mean(length)`. This tells the stats code to calculate the mean of the `length` enumerated value.

Stats operations can be *keyed*; for example, if you wish to calculate the count of log entries *from each Gravwell component*, the component names ("webserver", "indexer", "searchagent" etc.) are the *keys*. Keys are specified using the `by` keyword, as seen in the query below, which calculates the count of log entries and the mean Message length, per Gravwell component. Figure 4.16 shows the results for this query.

```
tag=gravwell syslog Appname Message | length Message
| stats count by Appname mean(length) by Appname | table Appname count
```

Bucketing and Condensing

When you've run a search using a statistics operation, you will frequently be able to zoom in on the results of a portion of the results (see Section 4.8.1 for a more detailed explanation), and the results will automatically update. For instance, consider the following query:

```
tag=gravwell stats count | table
```

If we run the query over the last hour, we'll see results as seen in Figure 4.17. We can then use the mouse to select a sub-portion of the timeframe graph at the top of the results, as seen in Figure 4.18; note how the count updates to reflect the count in the portion of time we selected, *without re-running the query.*

This is possible because the stats code uses bucketing and condensing operations. Basically, when the query runs, it generates a count *for each second*–we say it has a 1 second bucket size. However, because we want to display the full hour's count in our table, the *renderer* goes through and sums up all the count buckets to generate a final value. If you select a subset of the results, it is easy to compute the correct count by

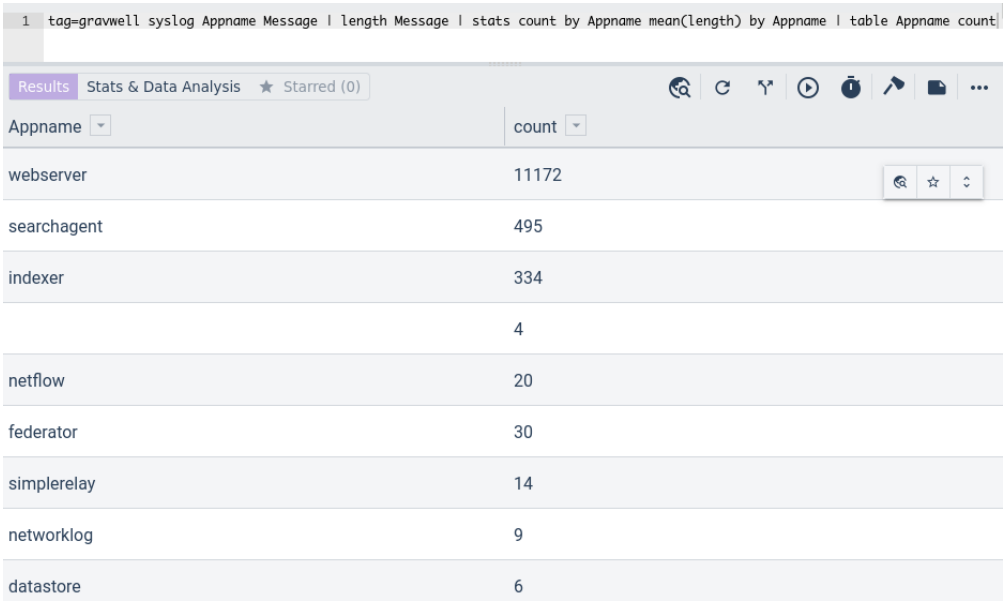


Figure 4.16: Keyed stats operation.

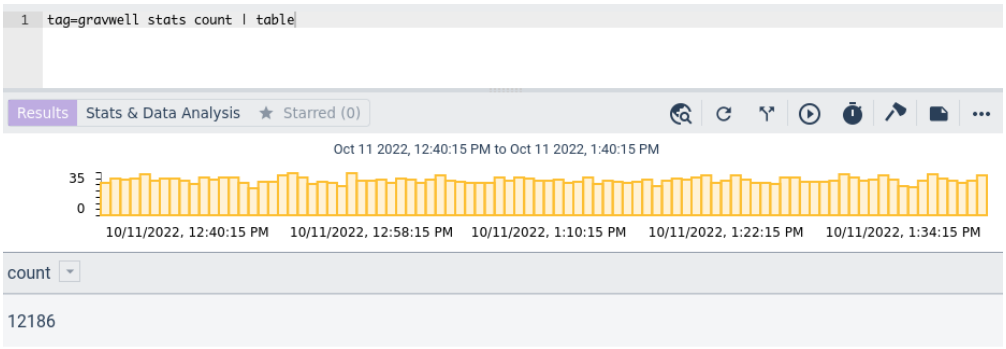


Figure 4.17: Un-zoomed query.

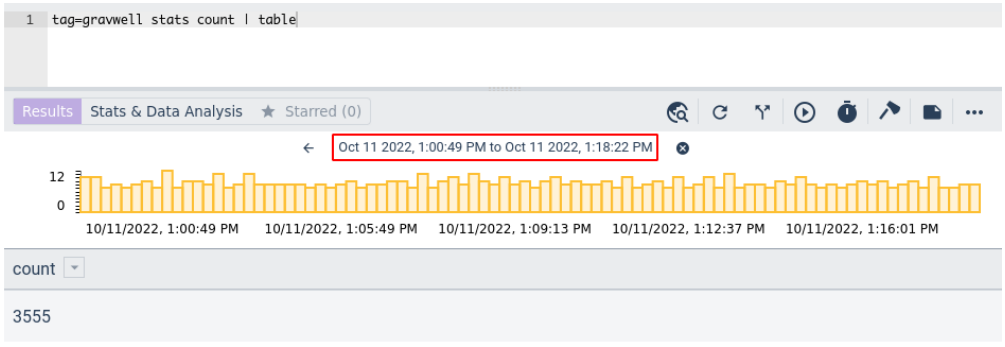


Figure 4.18: Zoomed query.

summing up only the buckets which fall within the selected timeframe. This same strategy is used for all the stats operations (we store additional data behind the scenes to accomplish this, e.g. in the case of `mean`).

If a search is executed over a very long period of time, the default bucket size may be adjusted. Consider that storing counts for each individual second on a query spanning 10 years would require hundreds of megabytes of storage!

It is also possible to manually specify a bucket size using the `over` keyword. Figure 4.19 shows an example of a chart showing *hourly* connection counts generated by simply counting the number of Zeek conn logs.

```
tag=zeekconn stats count over 1h | chart count
```

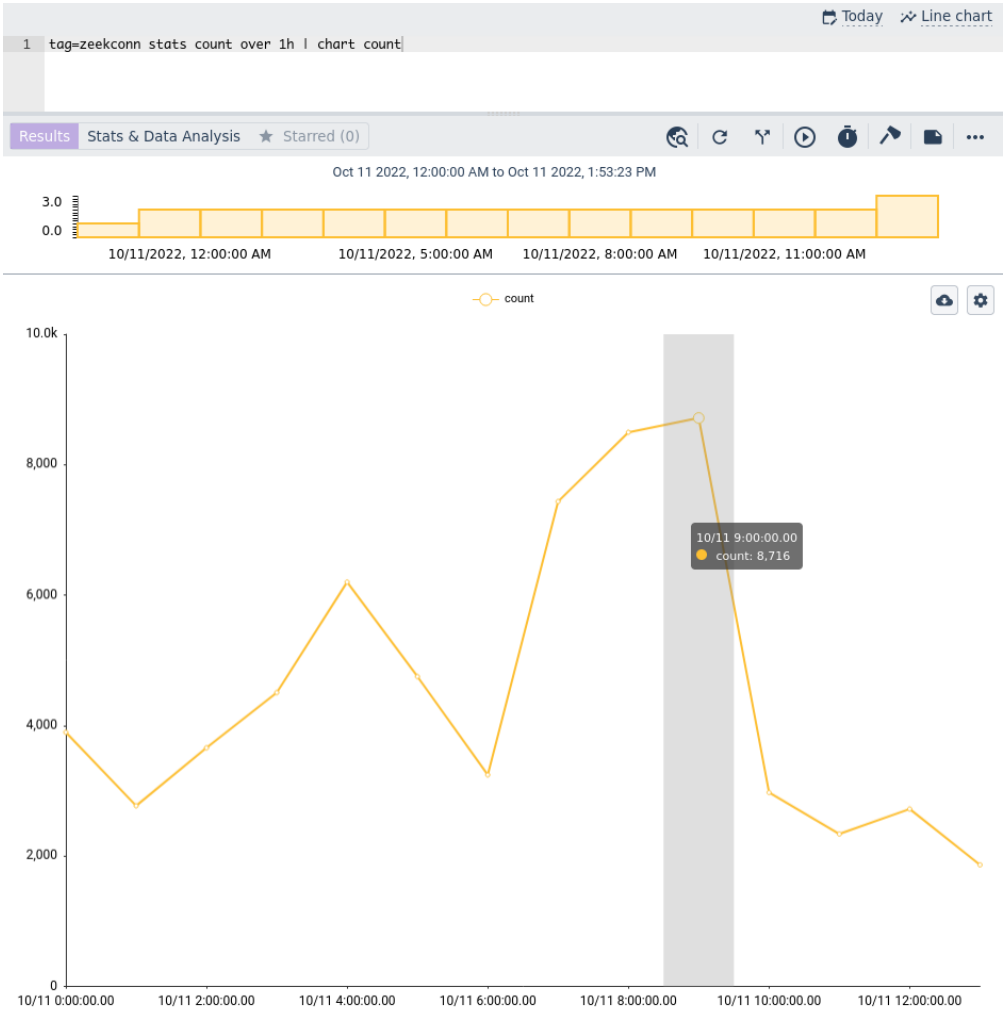


Figure 4.19: Using the over keyword.

Sum vs. Total

The sum and total operators behave exactly the same in every context except chart. When using the chart renderer the total module will not "reset" values across time buckets.

The behavior difference is best demonstrated using the following queries:

```
tag=zeekconn ax duration
| stats sum(duration) total(duration)
| table sum total
```

```
tag=zeekconn ax duration
| stats sum(duration) total(duration)
| chart sum total
```

When the sum and total values are viewed in the table renderer (Figure 4.20), the values are the same: the summed-up/totaled-up duration of connections from Zeek connection logs.

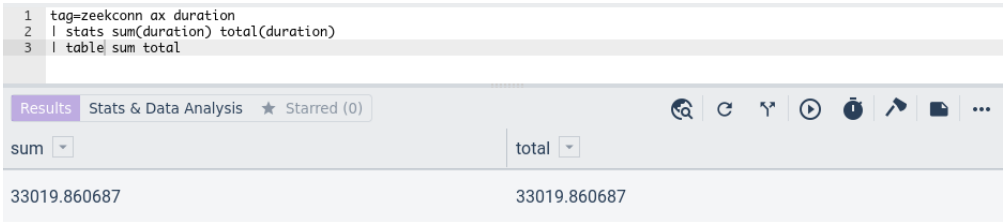


Figure 4.20: Sum and total operations in a table.

However, when viewed as a chart (Figure 4.21), we can see the difference. Each line is composed of dots representing a fraction of the total time frame. On the sum line, each dot represents the sum for that particular fraction of time. On the total line, each dot represents the total duration of that slice of time *and all preceding slices*. The underlying operations are the same, but total does not "reset" the bucket each time it transitions to a new time span, where sum does.

Statistics Shorthand Syntax

The sum, mean, max, min, count, variance, and stddev statistics operations can be invoked in "shorthand" syntax:

```
tag=* count
```

```
tag=gravwell length | mean length
\end{verbatim]

\begin{verbatim}
tag=gravwell syslog Appname | length | mean length by Appname | table Appname mean
```

Note that `stats mean(length)` becomes `mean length`. The shorthand syntax varies slightly from the normal syntax, because "shorthand" syntax is a compatibility mode remaining from old versions of Gravwell.

It is not possible to invoke multiple stats operations simultaneously using the shorthand syntax. In general, we recommend using the `stats` syntax, as it is less ambiguous and more flexible.

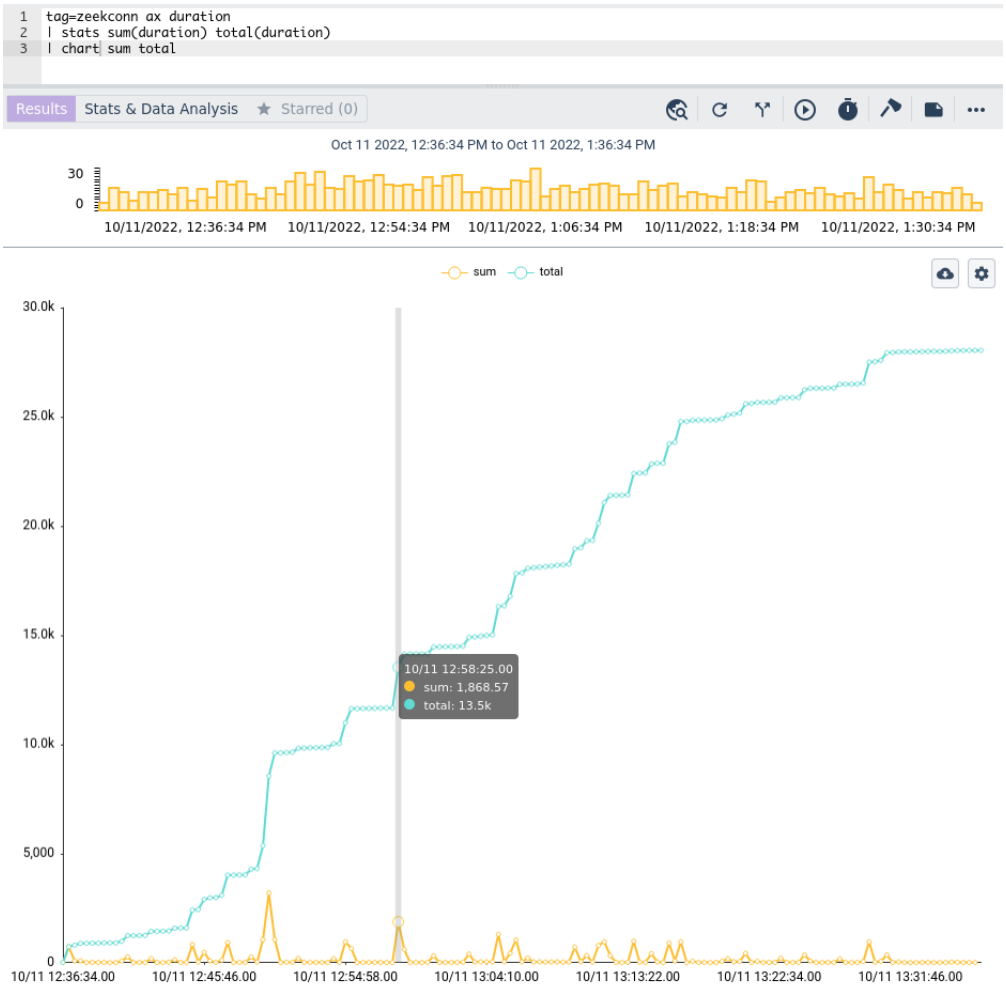


Figure 4.21: Sum and total operations in a chart.

4.8 Render Modules

Gravwell provides a selection of render modules to help users display their results in the most comprehensible manner possible. Render modules are in charge of receiving data from the search module pipeline and organizing it for display to the user. When possible, the renderers provide a second order temporal index, which allows the user to move around and zoom in on time spans within the original search. Renderers can optionally save search results, which can be reopened for later viewing or even passed to another instance of Gravwell. This is useful for archiving a view of data or saving the results even after the original data is expired or purposefully deleted.

4.8.1 Temporal vs. Non-Temporal Rendering

In discussing renderers, a distinction should be made between temporal and non-temporal mode. Most searches will operate in *temporal mode*, where the entries arrive at the renderer in order of their timestamp. This allows the renderer to display a timeline at the top of the results; the user can select portions of the timeline to zoom in on a particular subset of the results, as shown in Figure 4.22

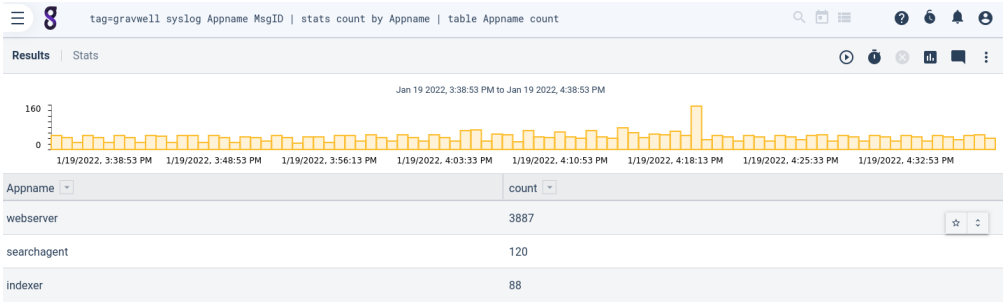


Figure 4.22: Renderer in temporal mode

If, however, the entries are sorted by something other than time, the renderer will enter *non-temporal* mode, in which subselection is not possible, as shown in Figure 4.23.

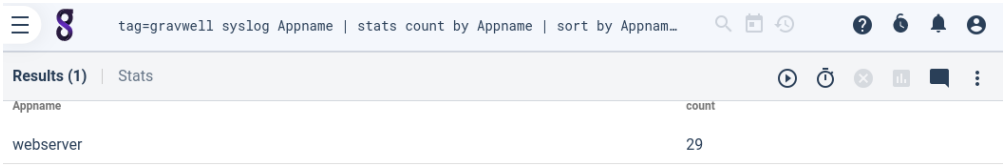


Figure 4.23: Renderer in non-temporal mode

4.8.2 Hands-on Lab: Temporal vs. Non-Temporal Rendering

In this hands-on lab we will use netflow records to identify the top communicating hosts both outbound and inbound from our network. First, launch a Gravwell container:

```
docker run --rm --net gravnet -p 8080:80 -d --name gravwell gravwell:base
```

Next, start the ingester container running the netflow ingester:

```
docker run --rm --net gravnet --name ingesters -it \
-e GRAVWELL_CLEARTEXT_TARGETS=gravwell:4023 \
gravwell:ingesters /opt/gravwell/bin/gravwell_netflow_capture
```

The netflow ingester is pre-configured to listen on port 2055 for incoming Netflow v5 records.

Now, we use another Docker container to generate Netflow records and send them to the ingester:

```
docker run -it --net gravnet --rm \
networkstatic/nflow-generator -t ingesters -p 2055
```

The netflow generator will run indefinitely, generating flow records, until killed.

Log into the web GUI (http://localhost:8080). We can now search the netflow data. Issue the following netflow search to convert the binary netflow data into a readable table of information. There are more fields available to netflow, but for this example we are only going to referring to some of them.

```
tag=netflow netflow Src Dst Bytes | table
```

Netflow provides records of communication flows to and from a host. The temporal overview chart shows bars indicating the presence of netflow records over time. Clicking and drag in this chart to zoom in on a spike or valley will cause Gravwell to display results within that time range (without re-issuing the search).

For this lab we're trying to answer the question of which hosts are egressing the most data and which are ingressing the most.

To answer the egress question, we can apply a filter to our netflow module that specifies non-routable private IP addresses. We want to sum the number of bytes sent by each private host to a public host.

```
tag=netflow netflow Src~PRIVATE Dst!~PRIVATE Bytes
| stats sum(Bytes) by Src | sort by sum desc | table Src sum
```

Observe that adding in the `sort by sum desc` component removes the overview chart from the top of the search results screen. This is because the search is now in non-temporal mode–we are sorting results by the sum, rather than by time.

To see which hosts are accumulating the most ingress traffic, we change the query slightly to invert the filtering of private address space in the netflow module, tell the stats module to sum by `Dst` instead of `Src`, and finally in table we specify `Dst` and the `sum`. Notice that the sort module does not change because it is operating on the enumerated value `sum`, which is created by stats regardless of the field being operated on.

```
tag=netflow netflow Src!~PRIVATE Dst~PRIVATE Bytes
| stats sum(Bytes) by Dst | sort by sum desc | table Dst sum
```

Note: The 'sort' module causes a search to "collapse" and may have performance impacts if used early on in a search pipeline. It should generally be the last module before the render module, which is 'table' during this lab. See the query optimization section (4.11) for more on this topic.

To clean up after the experiment, simply run:

```
docker kill $(docker ps -a -q)
```

4.8.3 Downloading Results

All renderers allow the user to download results in at least one format.

- Table: JSON, CSV, Lookup table (Gravwell native format).
- Raw/Text: JSON, CSV, text.
- Chart: JSON, CSV.
- Pointmap/Heatmap: JSON.
- FDG: JSON, CSV.
- Stackgraph: JSON.

The table renderer's CSV and lookup table options deserve some extra notes. Downloading a "lookup table" or CSV file from the table renderer results in a file which can be uploaded to Gravwell again as a resource and used with the lookup module; this is a convenient way to export a lookup table to another Gravwell system.

4.8.4 Text/Raw Renderers

These renderers provide only the most basic functionality but are useful when doing initial explorations on data. The text renderer is designed to show human readable entries in a text format, as seen in Figure 4.24. Any non-printable characters will be converted to the '.' character. Text also fully supports Unicode and can render non-ASCII characters. Text is the default renderer and is applied if no renderer is specified.

The raw renderer is functionally similar to the text renderer, but does not attempt to modify or change any non-printable characters. This renderer hands back the raw record, for better or worse. This renderer can be useful when passing data back to other tools which need the raw values, or when you just want to see if your browser can take a stab at turning packets into emojis.

These renderers have a default limit of approximately 1000 characters per entry, to prevent accidentally displaying multiple megabytes of raw data. To increase the maximum length of output add the limit argument, e.g. `text limit 2048` to display up to 2048 characters.

Hovering the mouse over an entry in the GUI when using either the text or raw renderer brings up a context menu with two options, as shown in Figure 4.25.

The right-most icon shows the built-in fields and any enumerated values on the entry, as seen in Figure 4.26. The star icon allows you to highlight particular entries; highlighted entries can be viewed later in the "Highlights" tab at the bottom of the page, as seen in Figure 4.27.

Figure 4.24: Text renderer entries view

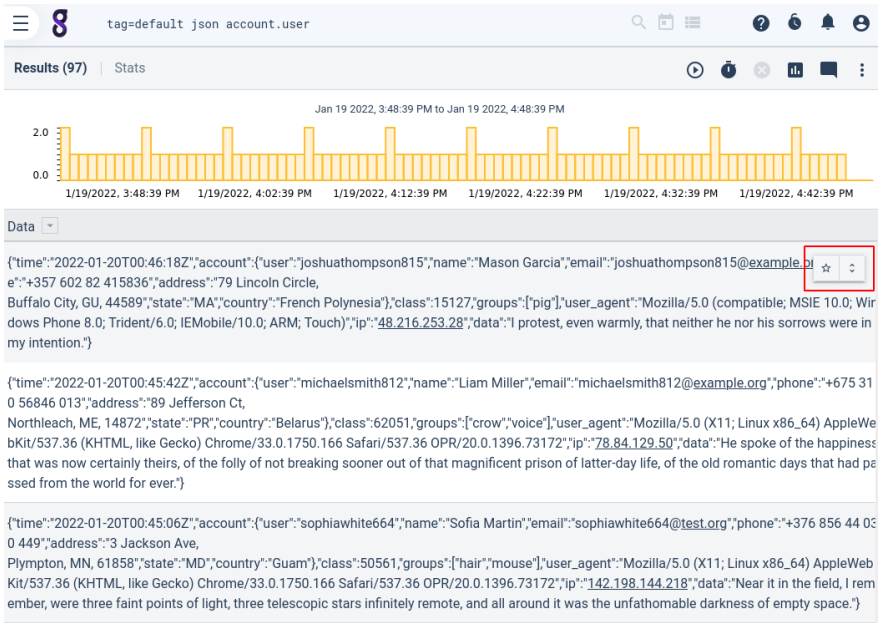


Figure 4.25: Text renderer context menu

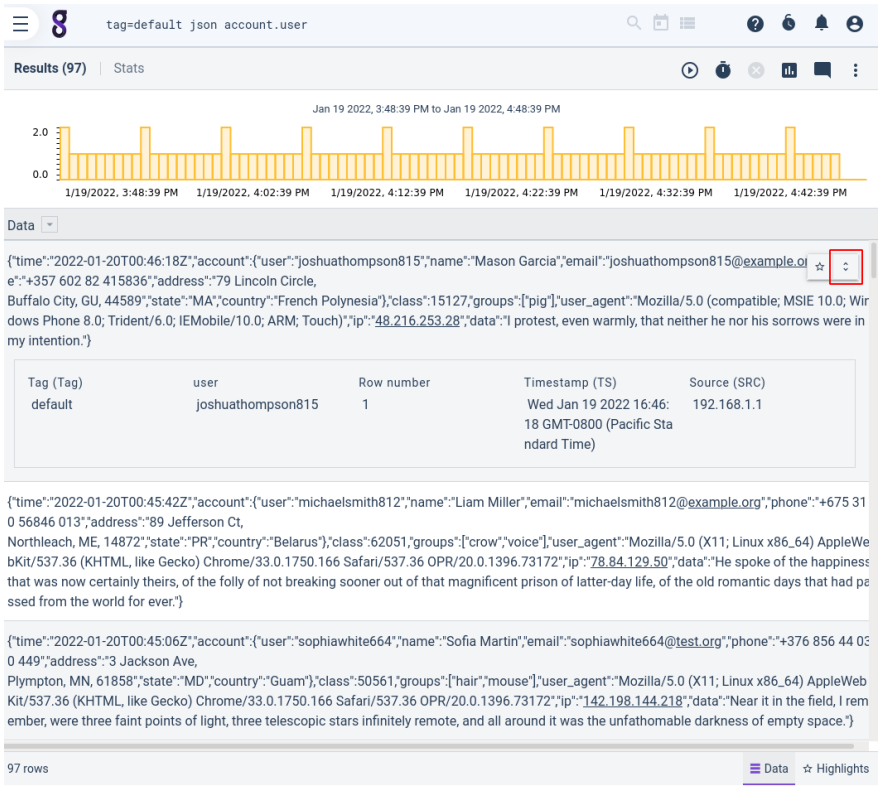


Figure 4.26: Text renderer enumerated values view

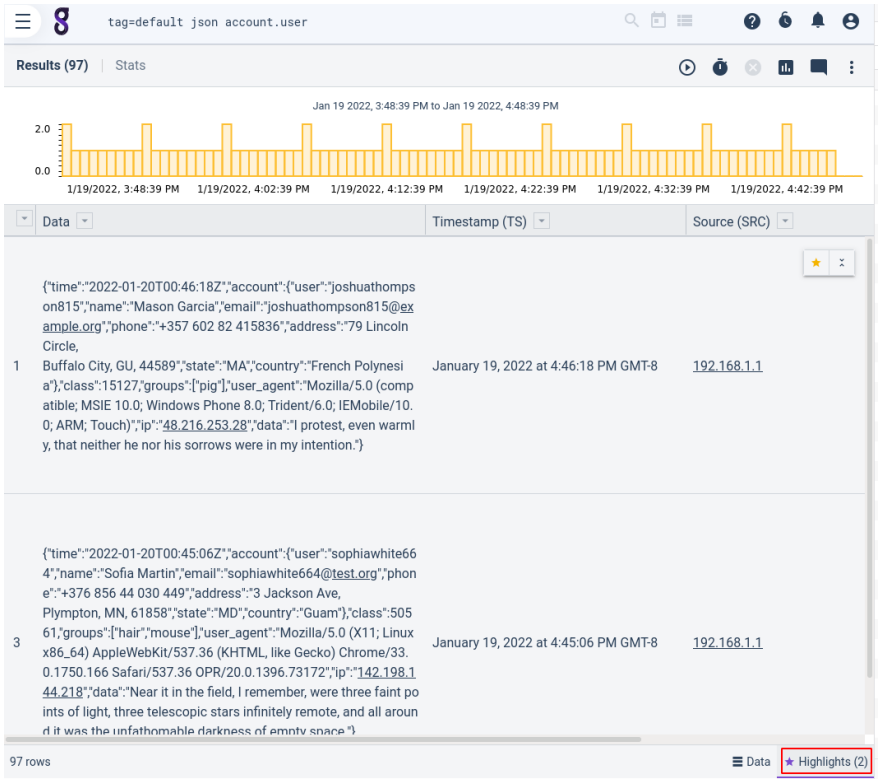


Figure 4.27: Text renderer highlight view

4.8.5 Table Renderer

The table renderer is used to create tables. The renderer's arguments specify which columns should be shown in the table. Arguments must be enumerated values, "DATA", "TAG", "TIMESTAMP", or "SRC".

Specifying no column arguments causes table to display all enumerated values as columns instead; this is useful for exploration.

Supported Options

- `-save <destination>`: save the resulting table as a resource for the lookup module. This is a useful way to save the results of one search (say, extracting a MAC→IP mapping from DHCP logs) and use it in later searches.
- `-nt`: Put the table into non-temporal mode. This causes upstream math modules to condense results rather than having table do it. This can seriously speed up searches over large quantities of data when temporal sub-selection is not needed.

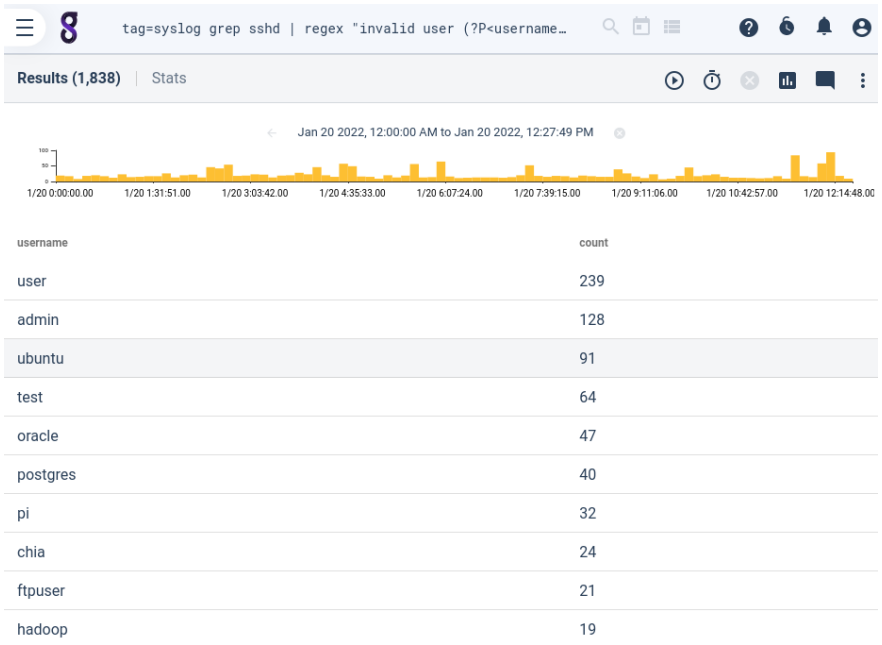
Basic table use

Extract a few elements from a Netflow record, then have table automatically display them:

```
tag=netflow netflow Src Dst SrcPort DstPort | table
```

Find failed SSH login attempts and count how many attempts were made for each username (Figure 4.28):

```
tag=syslog grep sshd
| regex "invalid user (?P<username>\S+)"
| stats count by username
| table username count
```



username	count
user	239
admin	128
ubuntu	91
test	64
oracle	47
postgres	40
pi	32
chia	24
ftpuser	21
hadoop	19

Figure 4.28: Failed SSH login table

Using the -save option

Use DHCP logs to build a lookup table containing IP to MAC mappings:

```
tag=syslog regex "DHCPACK on (?P<ip>\S+) to (?P<mac>\S+)"
| unique ip mac | table -save ip2mac ip mac
```

Then use the lookup table to find the MACs associated with SSH logins (results in Figure 4.29):

```
tag=syslog grep sshd
| regex "Accepted .* for (?P<user>\S+) from (?P<ip>\S+)"
| lookup -r ip2mac ip ip mac as mac |table user ip mac
```

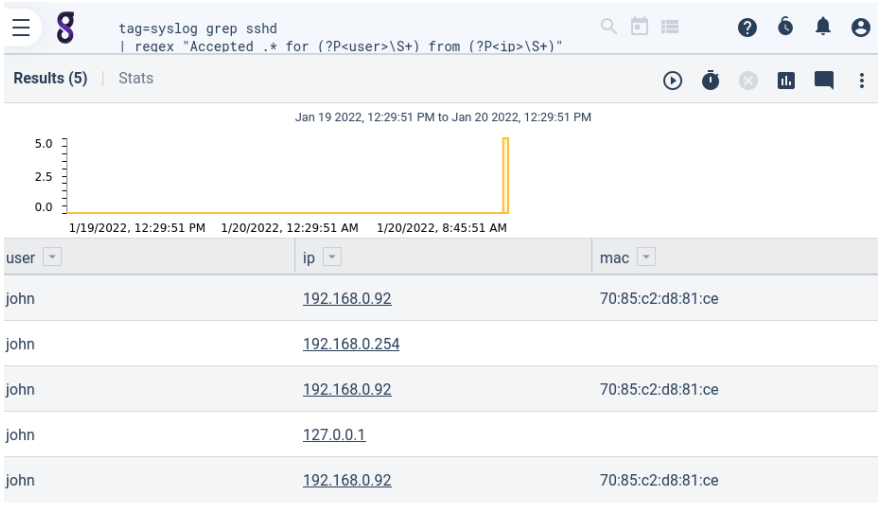


Figure 4.29: SSH login MAC addresses

Refer to the Automation chapter for a detailed description of how the `-save` flag can be used to automatically maintain lookup tables such as this.

Using the -nt option

In a situation with massive quantities of data, this flag will force table into non-temporal mode, making the `count` module condense results instead of table:

```
tag=jsonlogs json source | count by source | table -nt source count
```

4.8.6 Chart Renderer

The chart renderer is used display aggregate results such as trends, quantities, counts, and other numerical data. Charting will plot an enumerated value with an optional "by" parameter. For example, if there are counts associated with names, `chart count by name` will chart a line for each name showing the counts over time. The charting renderer will automatically limit the plotted lines or bar groups to 8 values. If you would like to see many more lines you can add the `limit <n>` argument, which tells the charting library to not introduce the "other" grouping until it exceeds the given limit of n values. The user interface for charting allows for a rapid transition between line, area, bar, pie, and donut charts.

The following query generates a chart showing the most common invalid usernames seen on incoming SSH connections–indicators of brute-forcing:

```
tag=syslog grep sshd
| regex "invalid user (?P<username>\S+)"
| stats count by username
| chart count by username limit 10
```

The "other" data group is a combined calculation of all remaining data sets that will not fit within the chart limit. In the example, we have instructed chart to only draw 10 groups by using the `limit 10` argument. This means that the chart module will first survey the data as it comes in to pick the 9 best data sets to include and then will add all other data sets to the "other" bucket. Chart calculates the value of the "other" data group using the exact same math as the rendered data sets, often leading to the "other" group dominating the output of a chart.

The results can be rendered in a variety of ways. By default, the UI will show a line chart (Figure 4.30), but the gear icon on the right side allows you to select a variety of chart types, including pie charts (Figure 4.31).

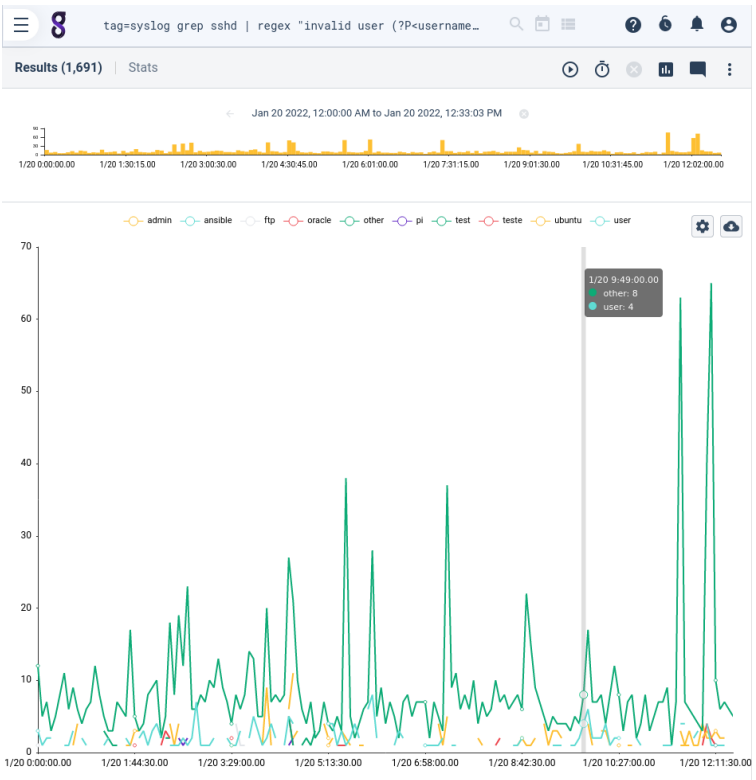


Figure 4.30: Line chart

Notice that the "other" group represents the largest slice in the pie chart, indicating that the vast majority

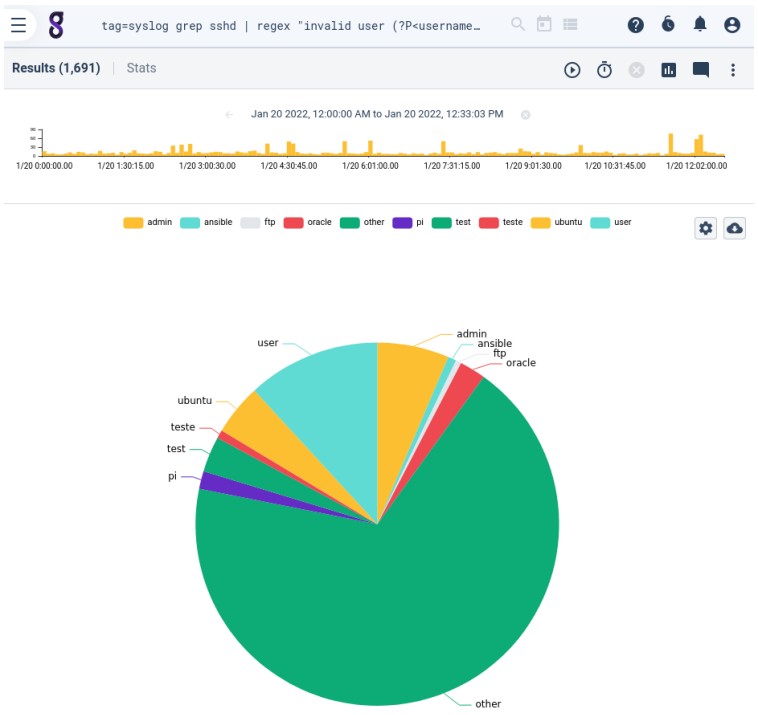


Figure 4.31: Pie chart

of usernames are not in the primary data series. To hide a group in the chart results, click its name in the legend at the top of the chart; this can be a useful way to amplify the "signal" above the "noise" of the "other" group.

Connect Null Values Setting

The "Connect null values" setting is used in line charts. By default, only data points which are very close together will be "connected", meaning that sparser data may end up looking like a collection of disconnected dots, as seen in Figure 4.32. Enabling this setting forces the renderer to join up the dots, as shown in Figure 4.33.

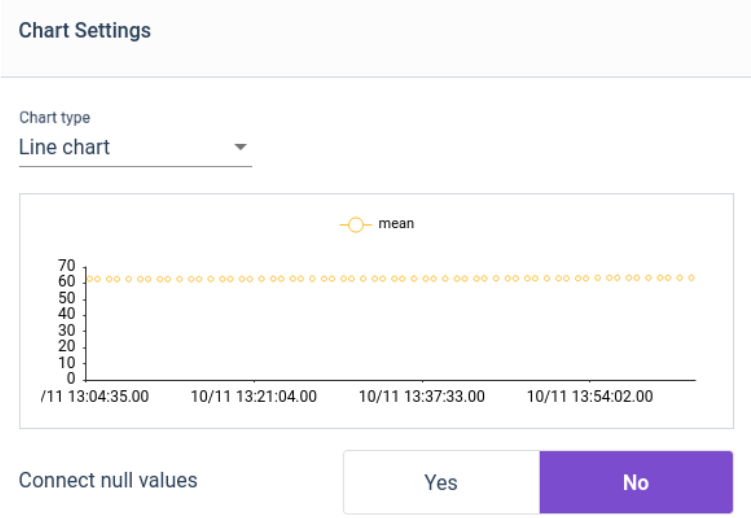


Figure 4.32: Line chart with "Connect null values" off.

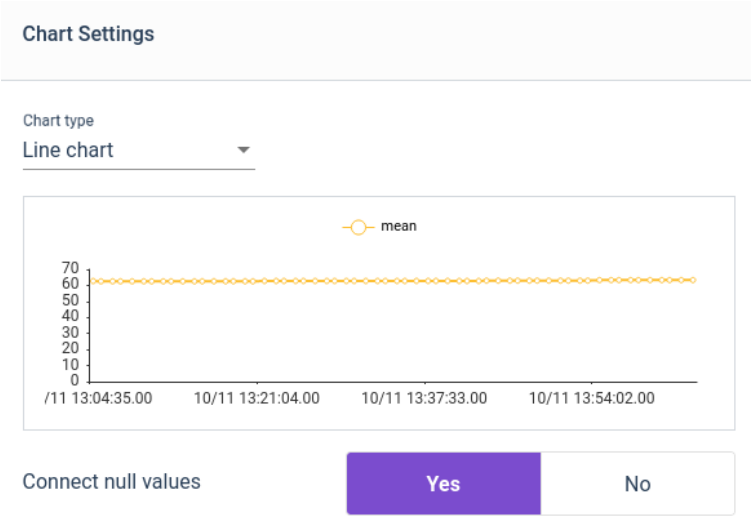


Figure 4.33: Line chart with "Connect null values" on.

4.8.7 Mapping Modules

The `pointmap` and `heatmap` renderer modules translate search results onto a map. Both place entries on the map based on locations in enumerated values. By default, the modules look for an enumerated value called 'Location', as set by the `geoip` search module. Locations can also be specified explicitly using the following:

- `-loc <enumerated value>` tells the module to look for a location in the specified enumerated value, rather than the default Location.
- `-lat <enumerated value> -long <enumerated value>` tells the module to look for the latitude and longitude values separately. These can be floating point numbers (as delivered by the geoip module) or strings from another source.

The map will display a maximum of 1000 points. It is *geofenced*, meaning that zooming in on one portion of the map will display up to 1000 points within that area.

Pointmap

Pointmap translates entries into distinct markers on the map. If additional enumerated value names are specified, their contents will be displayed when a point is clicked.

The following search displays a map of all IP addresses which connected to the SSH service, as seen in Figure 4.34.

```
tag=syslog grep sshd | regex "Connection from (?P<ip>\S+) port"
| geoip ip.Location | pointmap ip
```

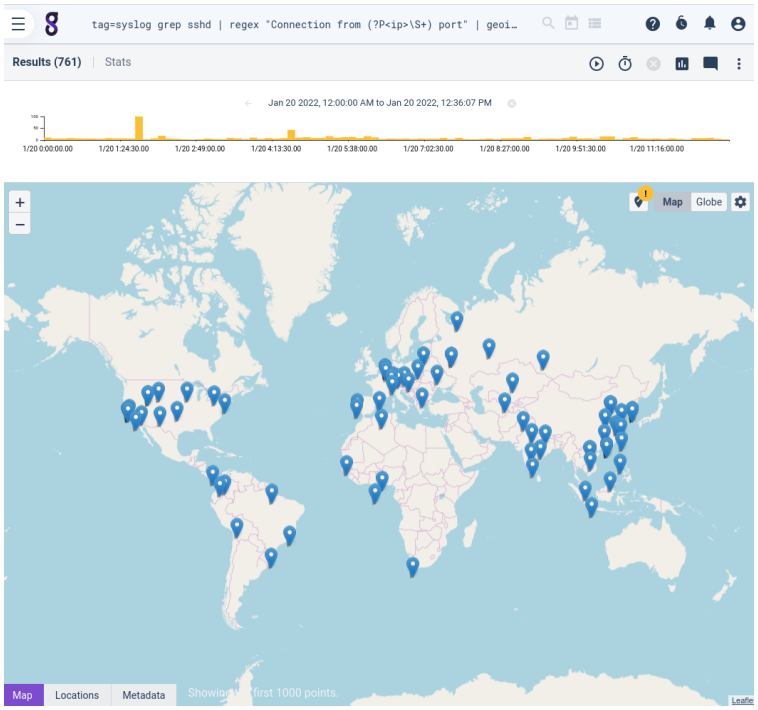


Figure 4.34: Pointmap IP locations

We can also use the geoip module to look up an IP address's ASN organization. Any enumerated values passed to the pointmap renderer can be viewed by clicking on a point, as seen in Figure 4.35

```
tag=syslog grep sshd
| regex "Connection from (?P<ip>\S+) port"
| geoip ip.Location
```

```
| geoip -r asn_db ip.ASNOrg
| pointmap ip ASNOrg
```

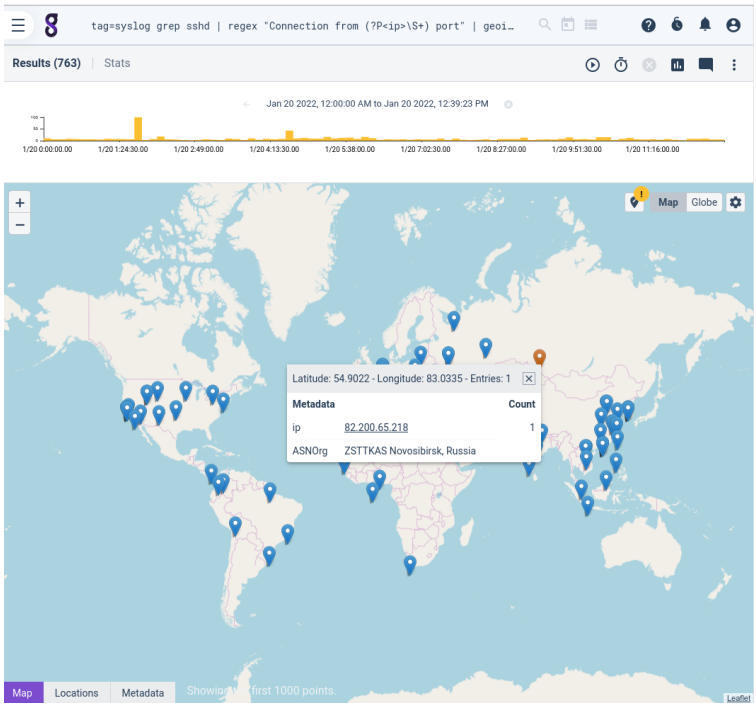


Figure 4.35: Pointmap with additional enumerated values

Heatmap

Heatmap operates similarly to pointmap, but it takes 0 or 1 additional enumerated values as arguments. If no enumerated value argument is given, it generates a heat map using the number of entries for each location as the 'heat'. In this example using IPFIX records, the 'heat' represents the number of connections from a location, as seen in Figure 4.36.

```
tag=ipfix ipfix ip!~PRIVATE
| geoip ip.Lat ip.Long
| heatmap -lat Lat -long Long
```

If we add the total number of bytes as an argument, the 'heat' is derived from the number of bytes sent over the connection, rather than the number of connections, as shown in Figure 4.37.

```
tag=ipfix ipfix ip!~PRIVATE bytes
| stats sum(bytes) by ip
| geoip ip.Location
| heatmap sum
```

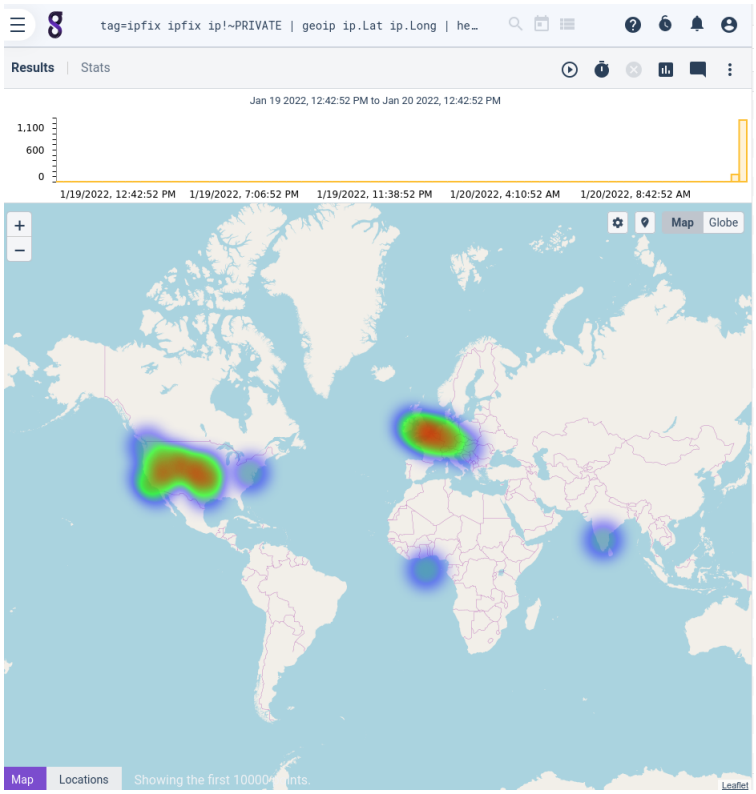


Figure 4.36: Basic heatmap use

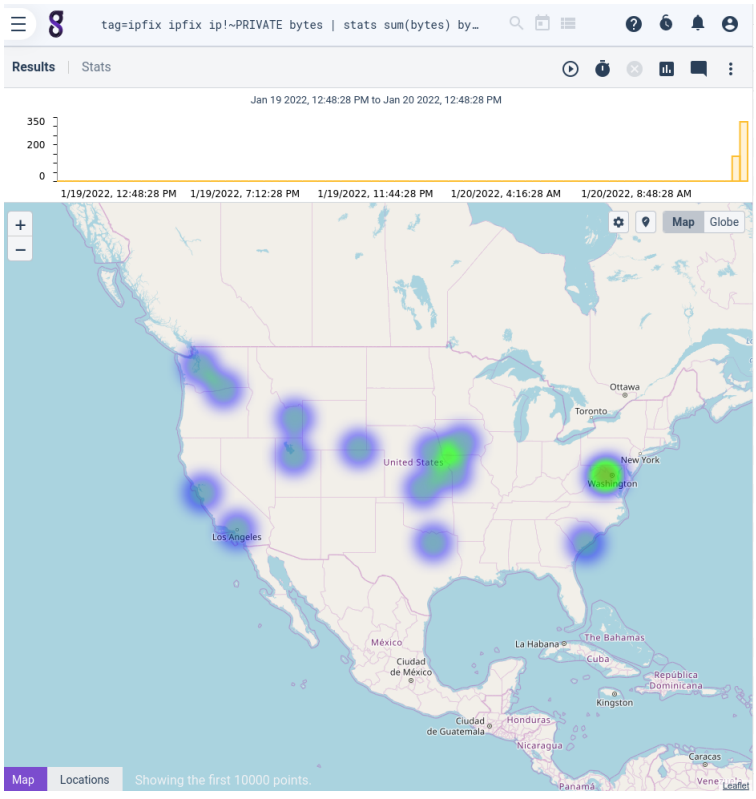


Figure 4.37: Heatmap using byte count as “heat”

4.8.8 Stackgraph Renderer

The `stackgraph` renderer is used to display horizontal bar graphs with stacked data points. A stackgraph is useful in displaying the magnitude of results that are accumulated from multiple components across a set of tags. The stackgraph renderer is an accumulator, meaning that it can interpret the operation of some upstream search modules and recalculate the results based on sub selections. In Gravwell terms, stackgraph supports second order searching and selection.

Stackgraph invocation requires three arguments which must be the names of enumerated values extracted by upstream search components. Argument one specifies the enumerated value which names each individual horizontal bar, for example an IP address. Argument two specifies the enumerated value which gives the individual components of each horizontal bar, for example a TCP port. Argument three is the magnitude value which represents the magnitude component of each stack value within a horizontal bar. Example magnitude components are count, sum, stddev, sum, max, and min. The easiest way to understand these arguments is by examining the examples below.

Note: Sorting data before sending it to stackgraph is unlikely to achieve the desired outcome. If you had a count of IP→port pairs and were interested in sorting based on that count and then sending to a stackgraph (e.g. `count by SrcIP,DstPort | sort by count desc | table SrcIP DstPort count`), then the first item in the list might be a port that has a very high count but only for one IP. For example, say port IP 10.0.0.1 spoke on port 443 with count 10000 but the next 8 entries are 8 different IPs all using port 80 with counts in the 9000 range, they will dwarf port 443 on the graph.

Stackgraph Example: Traffic Volumes by IP & Port

The following query will generate a stackgraph in which each bar respresents a single IP address, with components inside each bar representing the number of bytes sent to different TCP or UDP ports. See Figure 4.38 for a sample of the output.

```
tag=ipfix ipfix src ~ PRIVATE port < 1024 bytes
| stats sum(bytes) by src port
| stackgraph src port sum
```

Stackgraph Example: Failed SSH Logins by Country & User

This query will generate a stackgraph in which each bar represents a country and the components of the bars indicate the number of connections to a given port, based on IPFIX logs. Figure 4.39 shows the results.

```
tag=ipfix ipfix ip!~PRIVATE port < 1024
| geoip ip.Country
| stats count by port Country
| require Country
| stackgraph Country port count
```

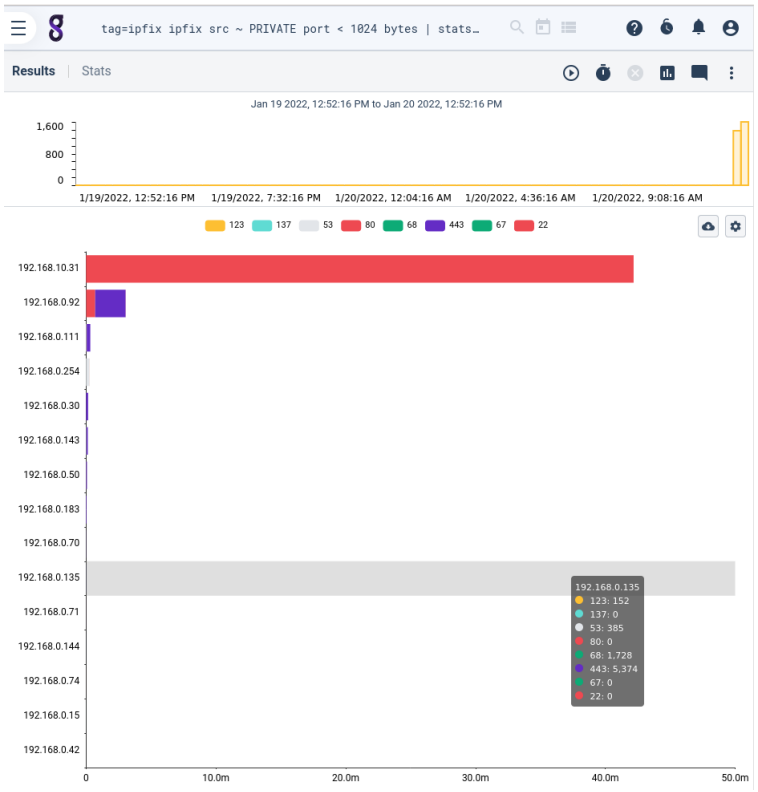


Figure 4.38: Stackgraph of traffic volumes by IP & Port

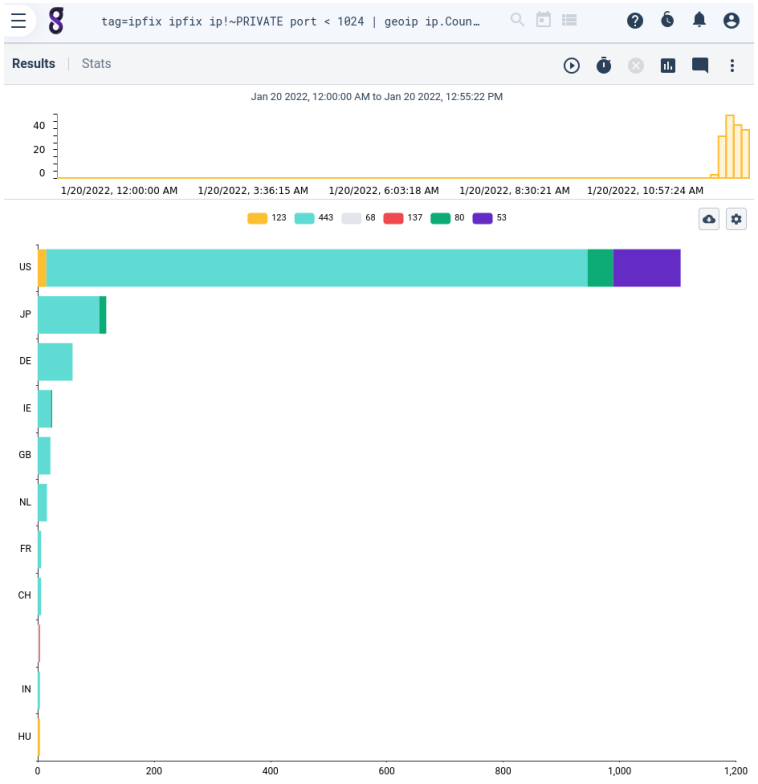


Figure 4.39: Stackgraph of connection count by country & port

4.8.9 Force-Directed Graph Renderer

The force-directed graph (fdg) module is used to generate a directed graph using node pairs and optional grouping. The fdg module accepts source and destination groups as well as a weight value for the resulting edge.

Supported Options

- `-b`: Indicates that edges are bidirectional, meaning that the pair [A, B] is equivalent to [B, A].
- `-v <enumerated value>`: Indicates that edges should be weighted as a sum of the provided enumerated value. The -v flag is useful in generating directed graphs where edges have weights represented by something other than a raw count.
- `-sg <enumerated value>`: Provides a group to apply to a source value which is used for coloring a graph. For example a source group may be a subnet for an IP which enables nodes in a graph to be grouped.
- `-dg <enumerated value>`: Same as -sg, but grouping based on destination parameter.

Examples

Use of the fdg module is easier to demonstrate than to explain. One example where a force directed graph can prove useful is to identify relationships between addresses on a network. Generating a weighted force directed graph of IPV4 traffic while grouping nodes into a class C network can be accomplished with the following query, as shown in Figure 4.40.

```
tag=pcap packet ipv4.SrcIP ipv4.DstIP ipv4.Length
| sum Length by SrcIP,DstIP
| subnet -t SrcSub SrcIP /24
| subnet -t DstSub DstIP /24
| fdg -v sum -sg SrcSub -dg DstSub SrcIP DstIP
```

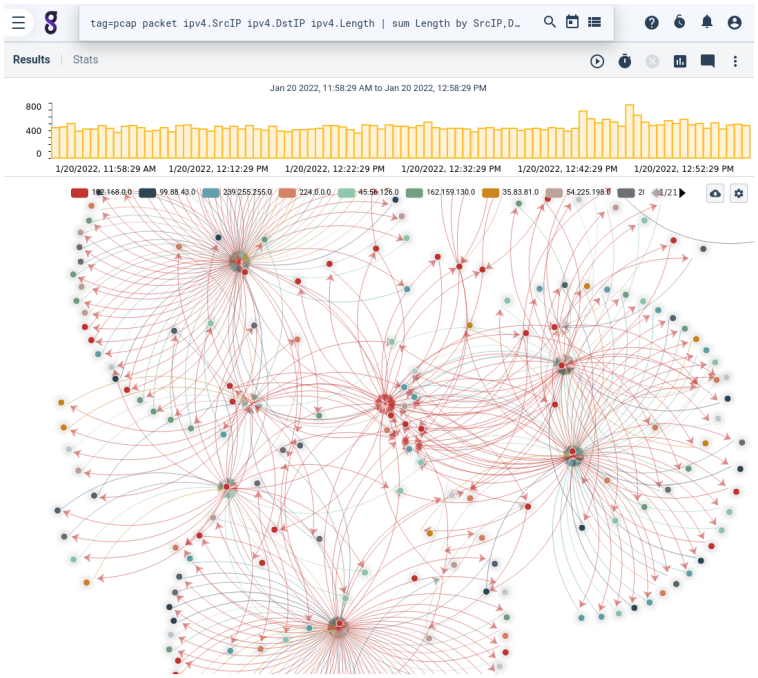


Figure 4.40: FDG of traffic between class C networks

Hovering the mouse over a node shows its label and the labels of its neighbors, as in Figure 4.41.

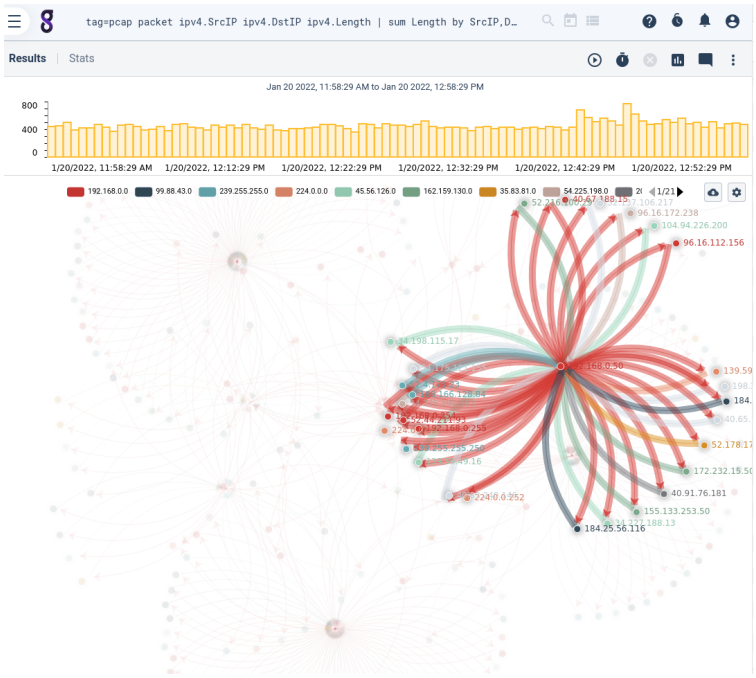


Figure 4.41: FDG context popup

4.9 Resources

Resources are persistent data objects which can be used in search queries. Resources can be manually uploaded by a user or automatically created by search modules. Resources are used by the lookup module to store lookup tables and by the anko module to store scripts.

The format of a resource is not restricted; from the point of view of Gravwell, a resource is simply a stream of bytes. Deriving meaning from that stream of bytes is up to the search modules: `lookup` expects data in a particular binary encoding, while `anko` simply treats the resource as a text file. Scripts written for the anko module may themselves create and access resources in a variety of formats, such as JSON-encoded text.

4.9.1 Resource Basics

Every resource is uniquely identified with a GUID, which is assigned when the resource is created. Resources also have a human-friendly name selected by the user. A resource can be accessed by specifying either the GUID or the name, but be aware that names can be changed. When building a dashboard or a search query you intend to share with others, we recommend using the GUID to refer to the resource.

Global resources are resources created by admin-level users for access by all users. Resources can also be shared with particular groups.

Resource data can be generated by hand or by running a search. For example, a search that results in a table display can be downloaded in the "lookupdata" format and uploaded into the resource system.

4.9.2 Resource name resolution

The resource system does not enforce unique resource names. Multiple users can have a resource named "foo", or indeed one user can own multiple resources named "foo". It is therefore important to be aware of the way the resource system resolves resource names into unique GUIDs.

Consider an example invocation of an anko script in a search: `anko myscript`. The resource manager will attempt to locate a resource named "myscript" in the following order:

- Check if the invoking user has a resource named "myscript"; if he or she has multiple resources with that name, it will return the first match.
- Check each group to which the user belongs. If there is a resource named "myscript" shared with one of the user's groups, it will return that resource.
- Check if there is a global resource named "myscript".

Note that the user could be a member of groups A and B, and that there could be one resource named myscript shared with group A and another unique resource named myscript shared with group B; which resource is returned is not certain. Similarly, if there are multiple global resources named myscript, any one of them could be returned.

This ambiguity can be overcome in two ways. The safest choice is to specify the resource as a GUID, which can be found in the resource management page, but GUIDs are very unwieldy and provide little useful context to the user. Luckily, it is also possible to select a resource by name with more precision by prefixing the resource name with a *namespace*. The following are valid namespace selections:

- `GLOBAL:myscript` specifies a global resource named myscript. This will ignore any resources owned by the invoking user and go straight to the global resources.
- `user=jfloren:myscript` specifies a resource named myscript belonging to the user jfloren. Note that this will fail if the invoking user does not have access to this resource.
- `group=security:myscript` specifies a resource named myscript which is shared to the group "security". Note that this will fail if the invoking user is not a member of the "security" group.

4.9.3 Managing resources with the GUI

Resources are managed via a page accessible from the main menu of the user interface. Open the menu and select "Resources" (Figure 4.42).

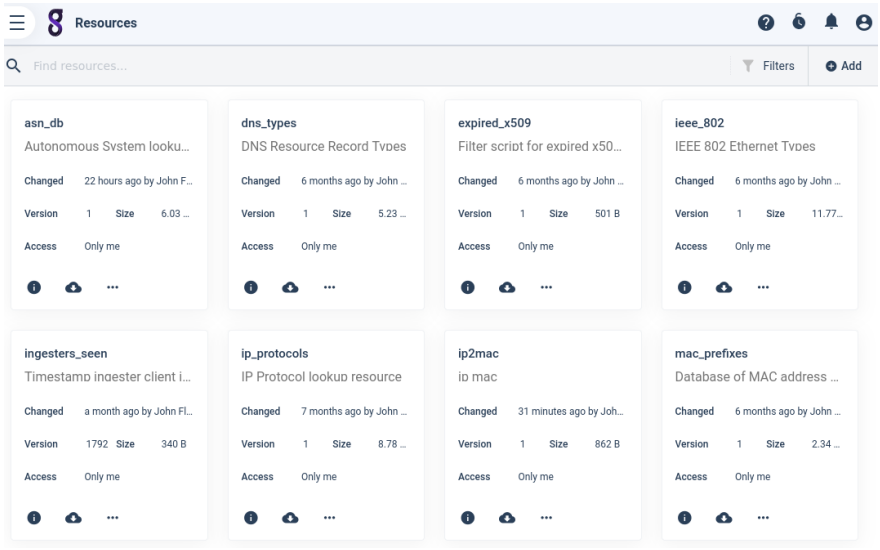


Figure 4.42: Resources management page

Resources can be created and deleted from this menu.

Deleting resources

To delete an existing resource, click the trash can icon next to the desired resource in the list.

Creating resources

To create a new resource, select the "Add" button in the upper right. This will open a dialog window, as shown in Figure 4.43. Set the resource name and description as desired and select any groups which should be able to read the resource, then select a file to upload. Note that the resource will not be created or uploaded unless the 'Save' button is clicked!

Editing resources

To edit an existing resource, click the pencil "Edit" icon below the desired resource in the resource list. This will open the resource editing screen as shown in Figure 4.44.

The name, description, and group sharing can all be managed from this screen. Admin users can also choose to make a resource global or non-global.

To change the actual contents of the resource, drag a file into the grey 'File' region or click to select a new file, exactly as when creating a new resource. Note that the Version, Hash, Size, and Last Modified fields change when a different file is uploaded.

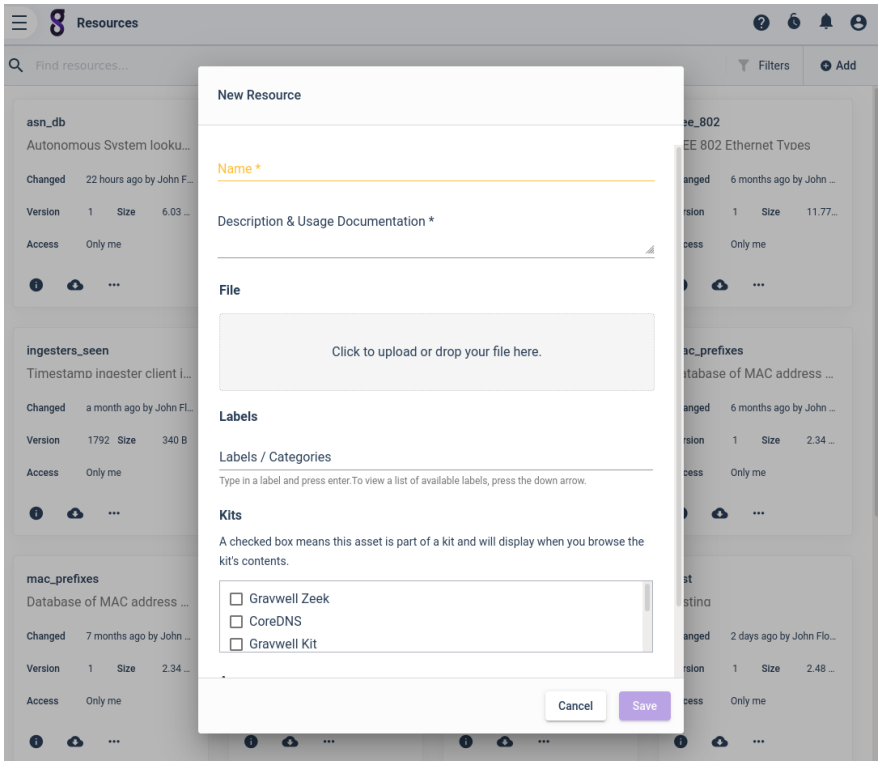


Figure 4.43: New resource dialog

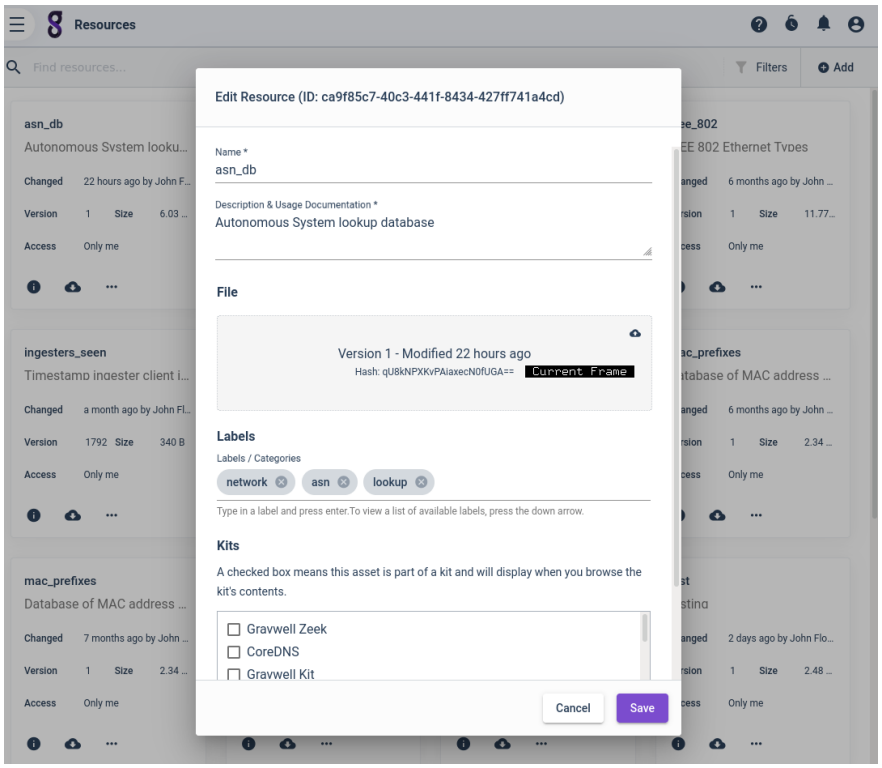


Figure 4.44: Resource editing dialog

4.9.4 Hands-on Lab: Enriching Netflow with GeoIP

This lab will demonstrate how resources can be used to enrich entries in the pipeline. We will use the geoip module to add location information based on the source IP of each flow, then show those flows on a map.

First, launch a Gravwell webserver+indexer container:

```
docker run --rm --net gravnet -p 8080:80 -d --name gravwell gravwell:base
```

Next, start the ingester container running the netflow ingester:

```
docker run --rm -d --net gravnet --name ingesters \
-e GRAVWELL_CLEARTEXT_TARGETS=gravwell:4023 \
gravwell:ingesters /opt/gravwell/bin/gravwell_netflow_capture
```

The netflow ingester is pre-configured to listen on port 2055 for incoming Netflow v5 records.

Now, we use another Docker container to generate Netflow records and send them to the ingester:

```
docker run -it --net gravnet --rm \
networkstatic/nflow-generator -t ingesters -p 2055
```

The netflow generator will run indefinitely, generating flow records, until killed.

Now, we will install a kit (Chapter 9) which contains the MaxMind GeoIP database. In the main menu, expand the "Kits" section, then click "Kit Management". This will open a list of currently-installed kits; if you see "Network Enrichment" in the list, you may skip on to the next step. If not, click "Available Kits" at the top of the page, then find the Network Enrichment kit and click the deploy icon, as seen in Figure 4.45. This will launch a kit deployment wizard; click through and install the kit.

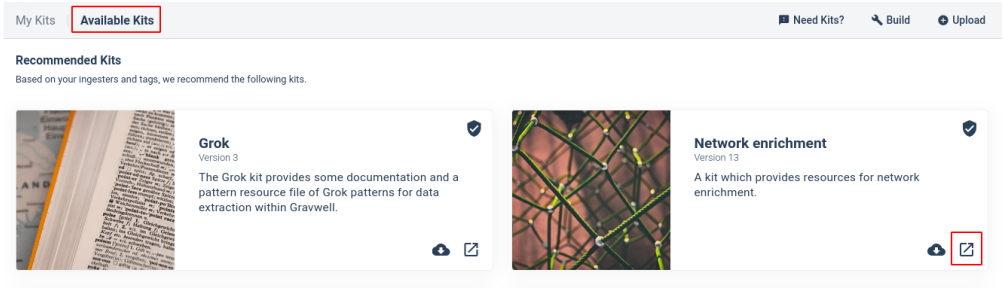


Figure 4.45: Installing the Network Enrichment kit.

With the kit installed, run the following query:

```
tag=netflow netflow Src | ip Src !~ PRIVATE
| geoip -r maxmind Src.Location | pointmap Src
```

This search pulls all entries tagged "netflow" and hands them to the netflow module, which extracts the Src field. The ip module is used to eliminate Src IPs in private subnets. The geoip module then uses the resource named "maxmind" to look up the Location for each Src IP address. Finally, the pointmap renderer draws each location on a map, as seen in Figure 4.46.

To clean up after the experiment, simply run:

```
docker kill $(docker ps -a -q)
```

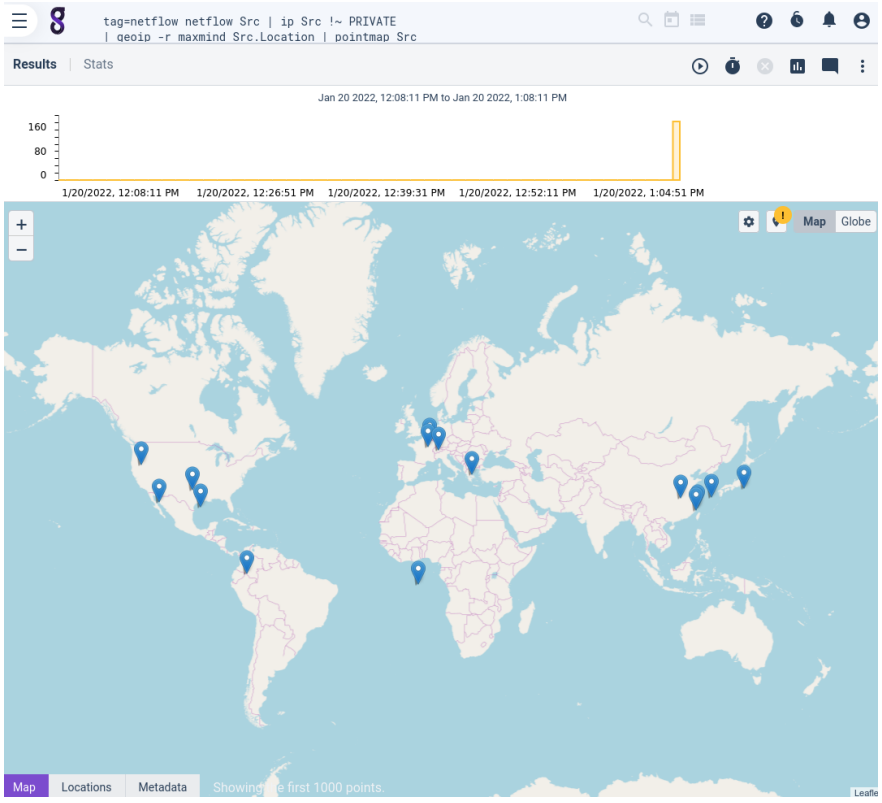


Figure 4.46: Netflow source IP pointmap

4.10 Data Fusion

The Gravwell query pipeline supports what we call module interleaving, which is basically the ability to specify that a given module should only process specific tags. This allows Gravwell to operate on multiple data formats at once and optionally fuse the results into a single cohesive data stream, essentially data fusion. Data fusion can be used to enrich one data stream from another, provide unified visibility across multiple data sources, or generate a single view for an operator that fuses many different data sources.

Tags form the basis of data control in the Gravwell pipeline. Previously we have only shown tags being used to control what data *enters* the pipeline. However, we can also add tag specifications to individual modules and to inform them that they should only process entries with those specific tags. Any entries that do not match the specified tags are passed through untouched. This is what we call data fusion.

Most data fusion queries are broken into three sections. The first section is the extraction phase where we use the data bypass and additional tag specifications to target modules against specific data types, for example we might have an unstructured log and JSON data. We use the tags to invoke the json module against the JSON data and the regex module against the unstructured data. The next section of a fusion query might use one data source to enrich or pivot on the other, potentially creating a running lookup system that adds additional enumerated values to one tag but derived from the other. The final section tends to look like regular search pipelines, using the data fused data without targeting any particular tag.

The following diagram shows what a data fusing query might look like. We have two data sources coming in, let's call them tag RED and tag BLUE. There are three modules that have been instructed to only operate on either RED or BLUE data, then there is a fusing module (orange) that uses the RED and BLUE data to create a single data stream with both data types fused. The final search module performs some standard operation on *all* entries before sending the data on to the renderer.

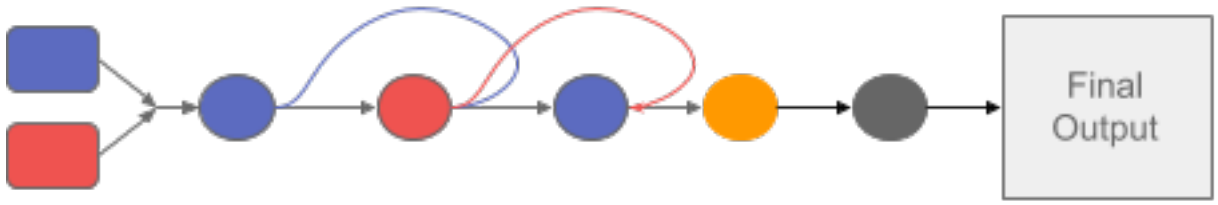


Figure 4.47: Data fusion notional pipeline

Let's look at an example query that represents this diagram. We have color coded the search module statements to help associate them with the diagram:

```
tag=RED,BLUE tag=BLUE json X Y
| tag=RED regex "(?P<B>\S+)\s(?P<X>\S+)\s"
| tag=BLUE regex -e X "[^\.]+\.(?P<B>.+)"
| stats sum(X) variance(Y) by B
| sort by sum desc
| table sum variance B
```

Data fusion queries that perform enrichment can use the `fuse` module in order to save enumerated values from one entry and attach them to another. Here is a less abstract query which uses dhcp (blue) logs to enrich dns (red) logs with MAC addresses:

```
tag=dns,dhcp tag=dhcp json mac ip
| tag=dns csv [0] as ip [1] as hostname
| fuse ip mac
| require -s hostname ip mac
| table hostname ip mac
```

This query looks daunting, but it is not terribly complicated. We have the `dhcp` and `dns` data streams coming in; in this case, we assume the DHCP logs are in a JSON format and the DNS records are CSV and use the appropriate modules to extract the enumerated values 'mac' and 'ip' from the DHCP logs, and 'ip' and 'hostname' from the DNS logs.

We then invoke the `fuse` module. When it receives a dhcp entry, it sees that the 'mac' enumerated value is set, so it stores the MAC address in a lookup table using the 'ip' as a key. When it sees a *dns* entry, it observes that there is no 'mac' enumerated value, so it checks the table to see if there is anything stored under the given 'ip'; if so, it sets the 'mac' enumerated value on the entry.

Consider these two example entries:

- Entry 1: `"mac": "00:11:22:33:44:55", "ip": "192.168.0.1"`

  TAG = "dhcp"

  mac = "00:11:22:33:44:55"

  ip = "192.168.0.1"

- Entry 2: `192.168.0.1,gravwell.example.org`

  TAG = "dns"

  ip = "192.168.0.1"

  hostname = "gravwell.example.org"

When the fuse module sees Entry 1, it will store a mapping of `192.168.0.1` → `00:11:22:33:44:55`. When it then sees Entry 2, it will observer that there is no 'mac' enumerated value set; it will therefore check its lookup table for anything stored with the key `192.168.0.1`. It finds the mapping stored from Entry 1 and thus creates an enumerated value on Entry 2, 'mac' = "00:11:22:33:44:55".

The `require` module simply serves to filter out any entries which were not able to be enriched, dropping any entry which does not have hostname, ip, and mac enumerated values on it.

4.10.1 Hands-on Lab: Data Fusion

For this lab we are going to be fusing data from three different sources: dhcp server logs and switch logs. The end result will be a single table that shows computers with their MAC address, IP address, hostname, and switch port. An admin can use this query to identify a machine by name and see the switch port is connected to, all in a single query. Let's start by ingesting our dataset and building extractions that pull the appropriate data from each tag. We will be using the tags "dhcp" and "switch".

Note that this lab is one of the most challenging in this training document. It makes use of the powerful but complex regex[3] and fuse[4] modules. We recommend carefully reading the examples shown in the lab instructions and those in the preceding section; these should demonstrate any necessary invocations of the modules. We also recommend using https://regex101.com/ to help build and test regular expressions if needed.

First, we'll start the Gravwell container:

```
docker run --rm --net gravnet -p 8080:80 -d --name gravwell gravwell:base
```

Then we'll ingest the data, found in the `gravwell_training/Search/Lab-Fusion` subdirectory:

```
cd ~/gravwell_training/Search/Lab-Fusion

docker run -v $PWD/data:/tmp/data --rm -i --net gravnet \
gravwell:ingesters /opt/gravwell/bin/reimport -rebase-timestamp \
-clear-conns gravwell:4023 -i /tmp/data/dhcp.json.gz -import-format json

docker run -v $PWD/data:/tmp/data --rm -i --net gravnet \
gravwell:ingesters /opt/gravwell/bin/reimport -rebase-timestamp \
-clear-conns gravwell:4023 -i /tmp/data/switch.json.gz -import-format json
```

Let's look at the switch tag first. Here is an example entry:

```
<135>2019-03-22 20:20:24 192.168.0.100 71565 The switch has learned a new
MAC address 50:2e:5c:e5:46:0b, vid:1024, interface:port 13
```

We want to extract the MAC address (50:2e:5c:e5:46:0b) and switch port (13). Let's use the following regular expression:

```
(?P<mac>[\da-f:]+),\s\S+\sinterface:port\s(?P<port>\d+)
```

Test your regular expression by running the query:

```
tag=switch grep "new MAC address"
| regex "(?P<mac>[\da-f:]+),\s\S+\sinterface:port\s(?P<port>\d+)"
| table mac port
```

Next let's extract our hostnames from the DHCP log. Here is an example entry where a client ACKs a DHCP lease; it contains the hostname, IP, and MAC address:

```
<30>1 2019-07-08T23:59:22.417203-06:00 router dhcpd 12365 - -
DHCPACK on 10.10.10.10 to e8:94:f6:1a:2b:c5 (keg) via insecure
```

Let's use the following regular expression to extract the IP, MAC, and potentially a hostname:

```
\s(?P<ip>[\d\.]+) to (?P<mac>[\da-f:]+)(\s\((?P<host>\S+)\))?
```

Test the regular expression with:

```
tag=dhcp grep "ACK on "
| regex  "\s(?P<ip>[\d\.]+) to (?P<mac>[\da-f:]+)(\s\((?P<host>\S+)\))?"
| table mac ip host
```

[3] https://docs.gravwell.io/#!search/regex/regex.md
[4] https://docs.gravwell.io/#!search/fuse/fuse.md

We now have the ability to extract all the relevant pieces from our tags, let's start combining them starting with the switch and dhcp tags. Let's look at a query that uses both tags to create a single data stream with ip, mac address, and switch port:

```
tag=switch,dhcp
tag=switch grep "new MAC address"
| tag=dhcp grep "ACK on "
| tag=dhcp regex "\s(?P<ip>[\d\.]+) to (?P<mac>[\da-f:]+)(\s\((?P<host>\S+)\))?"
| tag=switch regex "(?P<mac>[\da-f:]+),\s\S+\sinterface:port\s(?P<port>\d+)"
| table mac host port TAG
```

This should create a table that shows the extracted values for each tag. Now let's use eval to enrich the dhcp logs with the switch logs:

```
tag=switch,dhcp tag=switch grep "new MAC address"
| tag=dhcp grep "ACK on "
| tag=dhcp regex "\s(?P<ip>[\d\.]+) to (?P<mac>[\da-f:]+)(\s\((?P<host>\S+)\))?"
| tag=switch regex "(?P<mac>[\da-f:]+),\s\S+\sinterface:port\s(?P<port>\d+)"
| sort by time asc
| fuse mac port
| require host
| unique mac host port
| table mac host port
```

The `fuse` module in this case will store the port values from the switch logs and *set* them on dhcp logs with matching MAC addresses. The `require` module drops all entries which lack the 'host' enumerated value, which means all switch logs will be dropped. We then find all unique combinations of mac, host, and port, then generate a table.

Make note of the additional `sort by time asc` search module. Gravwell loosely orders data by time when executing queries, but correlations like DHCP and switch actions are operating on data where microseconds matter, so it is important to strictly order the data by time in the pipeline such that the switch logs properly occur *before* the DHCP logs. This way a MAC address from the DHCP log can be looked up against a MAC address from the switch log.

The resulting table (Figure 4.48) is pretty useful for keeping an eye on what devices are on your network. Note that some MAC addresses don't have a hostname associated with them; those are devices we may want to investigate!

Lab Tasks

1. Try adapting the query to also extract the vlan id (vid) from the switch logs and add it to the table.

   Hint: Look at the documentation for the fuse module to find out how to fuse multiple things!

2. Generate an FDG with host to vid
3. Can you find any hosts on more than one vlan?

Lab Questions

1. When would data fusion enrichment make more sense than creating a resource and using lookup?
2. When would data fusion not work
3. How would you combine the two methods to get "the best of both worlds"?

To clean up after the experiment, simply run:

```
docker kill $(docker ps -a -q)
```

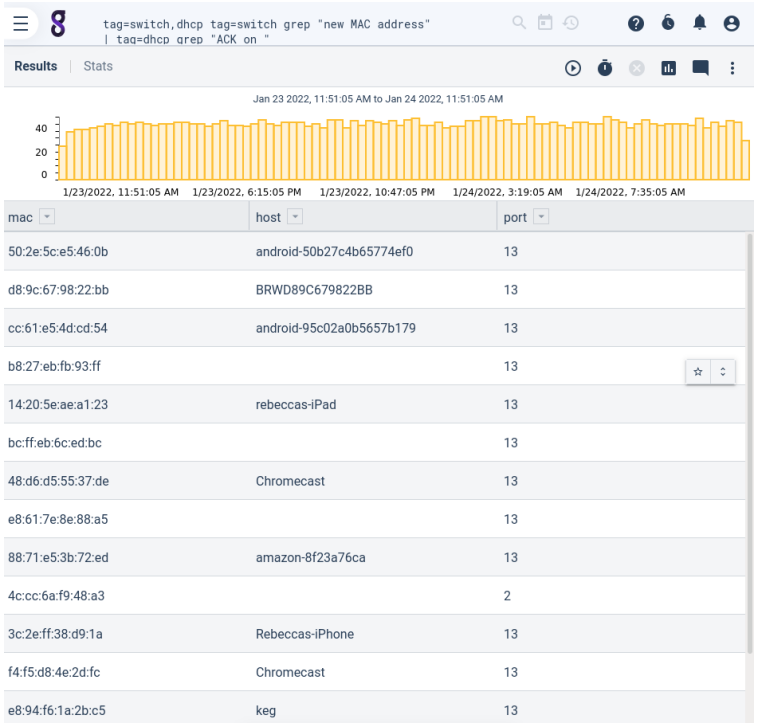


Figure 4.48: MAC / Hostname / Switch Port table

4.11 Query Optimization

The Gravwell query pipeline is very powerful. Searches are distributed to all nodes in the cluster, who intelligently share the load in order to maximize computing resources at top efficiency. It is possible, however, to issue searches in a way that degrades or hampers this. This section describes module ordering, condensing modules, and filtering and how to avoid common pitfalls.

For purposes of discussion, we are going to break modules down into three categories: parsers, operators, and condensers. Render modules are not a consideration when it comes to optimization, as they are always the final module in a pipeline.

4.11.1 Parsing modules

A parsing module is one that performs field extraction over a data entry. Typically, these modules are slower than operating modules as they usually read and process the entire data entry and create enumerated values for any given fields. The `json` module, as an example, will parse an entire JSON record and create enumerated values as directed.

For example, say we had some very large JSON entry that looked like:

```
{ 'field1': 'a', 'field2': 'aa', 'field3': 'aaa', 'field4': 'aaaa',
... 'field8000': 'aaaaaaaaaaaaaaaaaa....}
```

If we ran a search like `json field2 field488 field8000`, the json module would have to read and parse the entirety of the record. Gravwell distributes these parsing modules to every indexer in the cluster and these are run very close to the data.

4.11.2 Parsing modules and Accelerators

Accelerators are covered in Section 5.5 and in the online Gravwell documentation, but they should be mentioned when discussing query optimization. When turned on, they provide very powerful filtering speedups using our hybrid indexing technology. Invocation of those accelerators occurs in the parsing modules. In the case of no accelerators being present, filtering arguments that are provided in parsing modules are invoked after the parsing has occurred.

Using the above example, if we had accelerators turned on for "field3" and issued a search like: `json field3=="foo"`, this would invoke the accelerator framework to perform filtering before parsing of JSON is necessary. In this case, that parsing was done ahead of time by the accelerator and an index was created for rapid lookup.

In general, it is desirable to do field-based filtering in the parsing module, as it can engage acceleration if available.

4.11.3 Operator modules

For purposes of discussing optimization, operator modules are the common 'bread and butter' search modules available for the Gravwell search pipeline. They run in parallel close to source data (i.e. on the indexers). These modules are what do the filtering, extracting, enriching, and other analysis to be used further down the pipeline.

Note: Operator modules in pipeline *after* a condensing module will execute in series on the Gravwell webserver frontend.

4.11.4 Condenser modules

Condensing modules are those modules which require *all* of the data to be present. These modules trigger a collapse of the pipeline from a parallel series running on the indexers to a single pipeline running on the

Gravwell webserver. These are the modules that do counting, sum fields, strip non-unique values, etc. They require the entirety of data to be present in order to provide accurate results.

These modules are all of the math modules(count, sum, max, etc), stats, anko, and eval.

Let's use the query `json field1 | stats max(field1)` as an example where we are looking to find the maximum value of field1 in our data. When a collapse occurs, the indexers perform the analysis on what data they possess first, and then send the data to the Gravwell webserver to be aggregated in total.

Any modules following a condensing module will then be operating in series on the webserver, not in parallel on the indexers.

4.11.5 Hands-on Lab: Optimizing Queries

Let's look at a few searches that all accomplishing the same end results of filtering and analyzing some JSON data. We will execute these queries within the lab docker environments, but the differences won't be as noticeable on such a small scale. For any Gravwell deployment with significant data or node count, however, the knowledge in this section is invaluable.

We need to fire up the base Gravwell container and then the ingesters container to generate some sample JSON entries:

```
docker run --rm --net gravnet -p 8080:80 -d --name gravwell gravwell:base

docker run --net gravnet --rm -it --name jsoningesters gravwell:ingesters\
 /opt/gravwell/bin/jsonGenerator -clear-conns gravwell:4023 -entry-count 100000
```

Let's begin by examining the data:

```
tag=json
```

Here is an example of the data we're looking at:

```
{
   "time":"2022-06-09T13:03:36Z",
   "account":{
      "user":"jacobbrown802",
      "name":"Jacob Jones",
      "email":"jacobbrown802@example.org",
      "phone":"+850 169 94 122 338",
      "address":"58 Adams Circle, Ransom Canyon, PW, 97213",
      "state":"AZ",
      "country":"Netherlands Antilles"},
   "class":43294,
   "groups":["mouse","falcon"],
   "user_agent":"Mozilla/5.0 (Windows NT 6.3; Trident/7.0; rv:11.0) like Gecko",
   "ip":"102.115.187.35",
   "data":"One dog rolled before him, well-nigh slashed in half; but a second had him by
   ↪  the thigh, a third gripped his collar be- hind, and a fourth had the blade of the
   ↪  sword between its teeth, tasting its own blood."
}
```

The data generated is random. To complete this lab, examine the entries and make note of values found for user_agent, groups, and some random words found in data. For example, in our sample entry above, we may note the following:

- user_agent

  "Mozilla/5.0 (Windows NT 6.3; Trident/7.0; rv:11.0) like Gecko"

- groups

  "mouse"

  "falcon"

- random words found in data

  "dog"

  "collar"

  "sword"

In the searches below, make replacements using your pre-selected values.

Let's start with the base search that's looking to see how many entries for each group except 'mouse' have a specific user agent we are investigating and are not talking about a dog.

```
tag=json json -x groups groups account.user account.email ip data class user_agent
| eval user_agent=="Mozilla/5.0 (Windows NT 6.3; Trident/7.0; rv:11.0) like Gecko"
| grep -e data -v "[dD]og" | stats count by groups | grep -v -e groups "mouse" | table
↪  groups count
```

That query invokes a parsing module and then immediately condenses with an eval module that's inefficiently being used to match the user agent. We can improve this query by moving the user agent match into the parsing module:

```
tag=json json -x groups groups account.user account.email ip data class
user_agent=="Mozilla/5.0 (Windows NT 6.3; Trident/7.0; rv:11.0) like Gecko"
| grep -e data -v "[dD]og" | stats count by groups | grep -v -e groups "mouse" | table
↪  groups count
```

This query is also performing the check to remove the 'mouse' group after the pipeline has condensed. This filtering can also be moved into the parser module:

```
tag=json json -x groups groups!="mouse" account.user account.email ip data class
user_agent=="Mozilla/5.0 (Windows NT 6.3; Trident/7.0; rv:11.0) like Gecko"
| grep -e data -v "[dD]og" | stats count by groups | table groups count
```

This is looking a lot better but, depending on whether or not we have accelerators enabled for the various fields of this data, we can possibly improve the query even further. If accelerators are NOT enabled, it is actually more performant to do basic matching *before* the parsing module. The `grep` module is the fastest filtering module in the Gravwell pipeline when operating on the raw data entries. Let's improve further:

```
tag=json grep "Mozilla/5.0 (Windows NT 6.3; Trident/7.0; rv:11.0) like Gecko"
| json -x groups groups!="mouse" account.user account.email ip data class user_agent
| grep -e data -v "[dD]og" | stats count by groups | table groups count
```

We have optimized the query but beware, as this query now has potential to produce unintended results. If the data section of these log entries were to contain this `user_agent` string (perhaps a user is posting a message requesting technical support), then that entry would match the initial grep despite having an incorrect user agent. Thus, to further optimize for performance (and not for query length), we can put the filter back in the parsing module to ensure correctness of results:

```
tag=json grep "Mozilla/5.0 (Windows NT 6.3; Trident/7.0; rv:11.0) like Gecko"
| json -x groups groups!="mouse" account.user account.email ip data class
user_agent=="Mozilla/5.0 (Windows NT 6.3; Trident/7.0; rv:11.0) like Gecko"
| grep -e data -v "[dD]og" | stats count by groups | table groups count
```

Something to keep in mind is that if accelerators *are* enabled, then the initial grep would be a detriment to performance. See section 5.5 for additional details on query acceleration.

As an exercise, try and sort the above queries from worst to best by writing numbers in the margins. On the left margin, put the number ranking in a scenario where accelerators are disabled. On the right margin, put the number ranking where they are enabled.

To clean up after the experiment, simply run:

```
docker kill $(docker ps -a -q)
```

4.12 Auto-extractors

Gravwell enables per-tag extraction definitions that can ease the complexity of interacting with unstructured data and data formats that are not self-describing. Unstructured data often requires complicated regular expressions or slice specifications to extract desired data fields, which can be time-consuming to produce and prone to errors.

Auto-extractors are simply definitions that can be applied to tags which describe how to correctly extract fields from the data in a given tag. The "ax" module then automatically invokes the appropriate functionality of other modules. The auto-extractor system supports the following extraction methods:

- CSV
- Fields
- Regex
- Slice
- KV
- Syslog
- JSON
- Grok
- CEF

Auto-extractor definitions are used by the ax module which transparently references the correct extraction based on tags.

Note that only a single extraction can be defined per tag per user. Also note that auto-extractors always operate on the full underlying data of an entry. They cannot be used to perform extractions on Enumerated Values (the "-e" argument is disallowed)

4.12.1 Auto-Extractor Configuration

Every auto-extractor can define the following parameters:

- tag - The tag associated with the extraction
- name - A human-friendly name for the extraction
- desc - A human-friendly string that describes the extraction
- module - The processing module used for extraction (regex, slice, csv, fields, etc.)
- args - Module-specific arguments used to change the behavior of the extraction module
- params - The extraction definition

Only the module, params, and tag parameters are required; all others can be left blank if desired (for example, the regex module does not take any arguments, so args will always be empty).

4.12.2 Managing Auto-Extractors in the GUI

The Gravwell GUI can be used to manage extractors. The screenshot in Figure 4.49 shows the Extractors page with four defined extractors.

Note the buttons in the upper right. “Add” creates a new extractor interactively, allowing the user to enter appropriate values, as seen in Figure 4.50.

“Upload” allows you to directly upload files containing text versions of AX definitions; this is a convenient way to install many extractors at once.

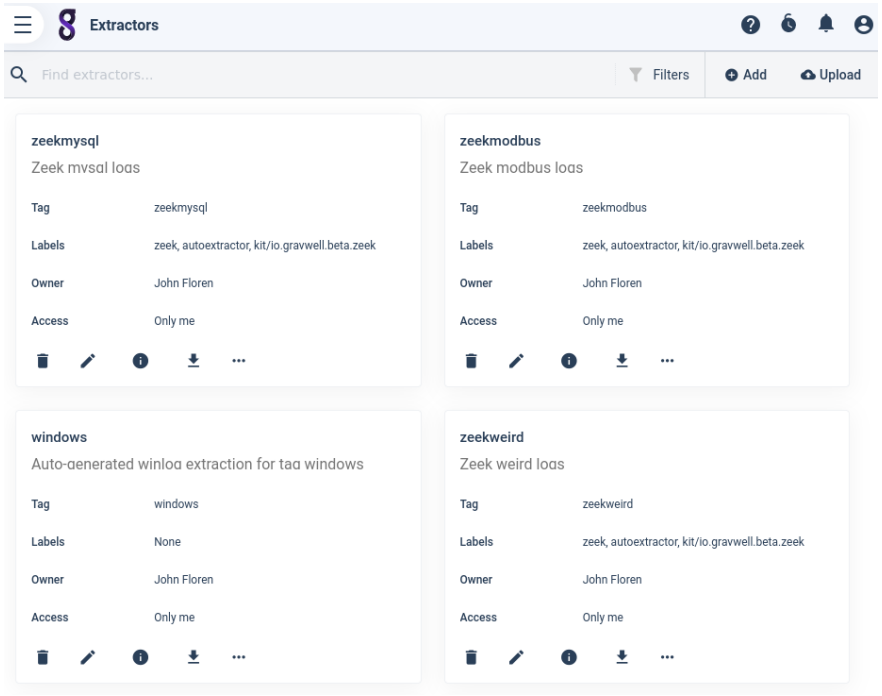


Figure 4.49: Extractors page

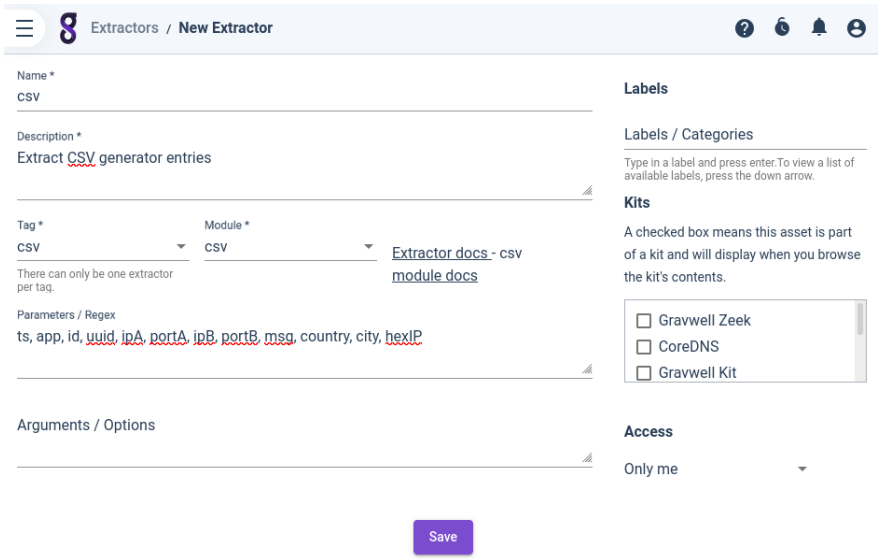


Figure 4.50: New extractor dialog

4.12.3 Auto-Extractor Files

Auto-extractors can be defined in text files, which may then be uploaded through the web interface. Auto-extractor files follow the TOML V4 format which allows comments using the "#" character. Each "ax" file can contain multiple auto-extraction definitions.

There are a few important rules about how the extraction parameters are defined in files:

1. Each extraction parameter's value must be defined as a string and double- or single-quoted.
2. Double-quoted strings are subject to string escape rules (pay attention when using regex), e.g. "\b" would be the backspace character (character 0x08) not the literal string "\b".
3. Single quoted strings are raw and not subjected to string escape rules, meaning '\b' is literally the backslash character followed by the 'b' character, not a backspace.

The ability to ignore string escape rules is especially handy for the "regex" processor as it makes heavy use of backslash.

Here is a sample auto-extraction file designed to pull some basic data from an Apache 2.0 access log using the regex module:

```
#pull ip, method, url, proto, and status from apache access logs
[[extraction]]
tag="apache"
name="apacheaccess"
desc="Apache 2.0 access log extraction to pull requester items"
module="regex"
args=""
params='^(?P<ip>\d+\.\d+\.\d+\.\d+)[^\"]+\"(?P<method>\S+)\s
(?P<url>\S+)\s(?P<proto>\S+)\"\s(?P<status>\d+)'
```

Multiple extractions can be specified in a single file by simply establishing a new `[[extraction]]` header and a new specification. Defining multiple extractions in one file is a convenient way to manage and share extractions across multiple Gravwell installations.

4.12.4 Extractor Examples

We will demonstrate a few auto-extraction definitions and compare and contrast queries which accomplish the same task with and without auto-extractors. We will also show how to use filters within AX.

In these examples, we show extractor definitions in the TOML file format described above; to instantiate them through the GUI, simply save them as a file and use the Upload button.

CSV

CSV or "Comma Separated Values" can be a relatively efficient text transport and storage system. However, CSV data is not self-describing, meaning that if all we have is a bunch of CSV data it can be difficult to tell what columns actually are. Auto-extractors can be used to predefine column names and make it dramatically easier to work with CSV data.

Here is an example data entry that is encoded using CSV:

```
2019-02-07T10:52:49.741926-07:00,fuschia,275,
68d04d32-6ea1-453f-886b-fe87d3d0e0fe,174.74.125.81,58579,
191.17.155.8,1406,"It is no doubt an optimistic enterprise.",
"TL",Bury,396632323a643862633a653733343a643166383a643762333a
```

There is a lot of data in there with no indication of which fields are what. To make matters worse, CSV data can contain commas and surrounding spaces which makes identifying columns with the naked eye very difficult. Auto-extractors allow the administrator (or user) to identify column names and types *once*; once defined, users can transparently leverage them using the "ax" module.

The following query manually extracts and names each element:

```
tag=csvdata csv [0] as ts [1] as name [2] as id [3] as guid [4] as src
[5] as srcport [6] as dst [7] as dstport [8] as data [9] as country
[10] as city [11] as hash
| table
```

With the following auto-extractor configuration installed:

```
[[extraction]]
  name="testcsv"
  desc="CSV auto-extraction for the super ugly CSV data"
  module="csv"
  tag="csvdata"
  params="ts, name, id, guid, src, srcport, dst, dstport, data, country, city, hash"
```

That same query becomes:

```
tag=csvdata ax | table
```

Note: The CSV auto-extraction processor does not support any arguments. The position of the names in the params variable indicates the field name. Treat it as a CSV header.

Fields

The fields module is an extremely flexible processing module that can define arbitrary delimiters and field rules in order to extract data. Many popular security applications like Bro/Zeek default to TSV (tab separated values) for data export. Other custom applications may use weird separators like "|" or a series of bytes like "//". The fields extractor can handle them all, and when combined with auto-extractors users don't have to worry about the details of the data format.

Unlike other auto-extractor processors, the fields module has a variety of configuration arguments. The list of arguments is fully documented in the fields module documentation. Only the "-e" flag is unsupported.

First, consider this tab delimited entry (line-wrapped in this document for readability):

```
2019-02-07T11:27:14.308769-07:00    green    21.41.53.11    1212
57.27.200.146    40348    Have I come to Utopia to hear this sort of thing?
```

Using the fields module to extract each data item, the query would be:

```
tag=tabfields fields -d "\t" [0] as ts [1] as app [2] as src [3] as srcport
[4] as dst [5] as dstport [6] as data
| table
```

An auto-extraction configuration to accomplish the same thing is:

```
[[extraction]]
    tag="tagfields"
    name="tabfields"
    desc="Tab delimited fields"
    module="fields"
    args='-d "\t"'
    params="ts, app, src, srcport, dst, dstport, data"
```

Using the ax module and the configuration above, the query becomes:

```
tag=tagfields ax | table
```

The following entry uses the more unusual "|" separator:

```
2019-02-07T11:57:24.230578-07:00|brave|164.5.0.239|1212|
179.15.183.3|40348|"In C the OR operator is ||."
```

Note that the last field contains the delimiter. The system that generated this data knew that it needed to include the delimiter in a data item, so it encapsulated that data item in double quotes. The fields module knows how to deal with quoted data; specifying the "-q" flag will make the module respect quoted fields. The quotes are kept on the extracted data unless the "-clean" flag is also specified.

Using the fields search module alone, the query would be:

```
tag=barfields fields -d "|" -q -clean [0] as ts [1] as app
[2] as src [3] as srcport [4] as dst [5] as dstport [6] as data
| table
```

But with an appropriate auto-extraction configuration (shown below) the query can still be the extremely simple `tag=barfields ax | table`:

```
[[extraction]]
    tag="barfields"
    name="barfields"
    desc="bar | delimited fields with quotes and cleaning"
    module="fields"
    args='-d "|" -q -clean'
    params="ts, app, src, srcport, dst, dstport, data"
```

Regex

Regex may be the most common use for auto-extractors. Regular expressions are hard to get right, easy to mistype, and difficult to optimize. Defining an auto-extractor allows the Gravwell administrator to define a regex once and take the burden off the users.

Here is an example entry with a very chaotic format (which is not uncommon in custom application logs):

```
2019-02-06T16:57:52.826388-07:00 [fir] <6f21dc22-9fd6-41ee-ae72-a4a6ea8df767>
783b:926c:f019:5de1:b4e0:9b1a:c777:7bea 4462 c34c:2e88:e508:55bf:553b:daa8:59b9:2715
557 /White/Alexander/Abigail/leg.en-BZ Mozilla/5.0 (Linux; Android 8.0.0;
Pixel XL Build/OPR6.170623.012) AppleWebKit/537.36 (KHTML, like Gecko)
Chrome/60.0.3112.107 Mobile Safari/537.36 {natalieanderson001@test.org}
```

The data is a really ugly access log for some custom application. The components of the entry are:

- ts - the timestamp at the beginning of each entry
- app - a string representing the handling application
- uuid - a unique identifier
- src - source address, both IPv4 and IPv6
- srcport - source port
- dst - destination address, both IPv4 and IPv6
- dstport - destination port
- path - URL like path
- user_agent - useragent
- email - email address associated with the request

Here is an extractor definition which can pull out those fields via regex:

```
[[extraction]]
module="regex"
tag="test"
params='(?P<ts>\S+)\s\[(?P<app>\S+)\]\s<(?P<uuid>\S+)>\s(?P<src>\S+)\s
(?P<srcport>\d+)\s(?P<dst>\S+)\s(?P<dstport>\d+)\s(?P<path>\S+)\s
(?P<user_agent>.+)\s\{(?P<email>\S+)\}$'
```

In order to extract all those fields using only the regex search module, the user would have to run the following:

ID	Timestamp seconds	Timestamp nanoseconds	Temperature (32bit float)	ASCII name
bits 0:2	bits 2:10	bits 10:18	bits 18:22	bits 22:

Table 4.5: Binary data structure

```
tag=test regex "(?P<ts>\S+)\s\[(?P<app>\S+)\]\s<(?P<uuid>\S+)>\s
(?P<src>\S+)\s(?P<srcport>\d+)\s(?P<dst>\S+)\s(?P<dstport>\d+)\s
(?P<path>\S+)\s(?P<user_agent>.+)\s\{(?P<email>\S+)\}$"
| table
```

However, with the auto-extractor and the ax module, the search is much simpler:

```
tag=test ax | table
```

To filter on a field using the ax module, simply attach a filter directive to the named field on the ax module call. This example will show all entries that have "test.org" in the email address while still rendering a table with all extracted fields:

```
tag=test ax email~"test.org" | table
```

To only extract specific fields, specify those fields as arguments to the ax module. This directs the ax module to only extract those specific fields, rather than extracting all fields by default:

```
tag=test ax email~"test.org" app path | table
```

Slice

The slice module is a powerful binary-slicing system that can extract data directly from binary data streams. Gravwell engineers have developed entire protocol dissectors using nothing but the slice module. However, cutting up binary streams of data and interpreting the data is not for the faint of heart, and slice module queries can be time-consuming to construct and understand. The slice extractor reduces this cognitive load for the user.

Showing binary data in text form is difficult, so in this document data is represented in hex encoding. These examples will operate on a binary data stream coming from a small control system that regulates a refrigerant compressor to maintain precise temperature control in a brewing system. The control system ships strings, integers, and some floating point values, and as is often the case in control systems all the data is in Big Endian order.

Note: The slice AX processor does not support any arguments (e.g. no "-e" allowed)

The slice AX processor is designed to cast data to specific types. As such its filtering options are a little more nuanced than other modules. Each extracted value has a specific set of filter operators based on its type. For a full description of filtering operators and types, see the slice module documentation.

Passing the example entries through the hexlify module:

```
tag=keg hexlify
```

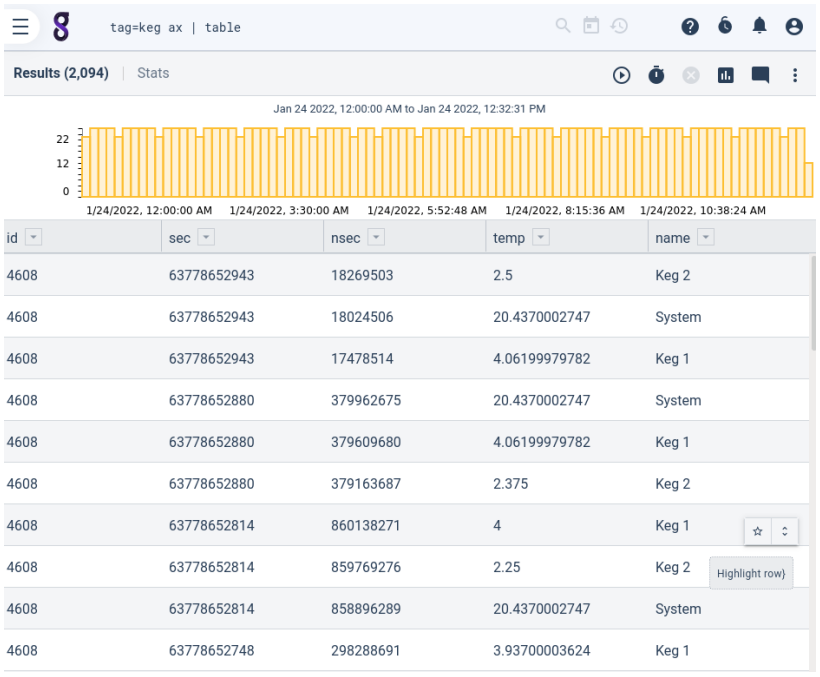
Results in output that look like this:

```
12000000000ed3ee7d4300000000014de536401800004b65672031
```

With some investigation, the packed binary structure was found to contain the structure shown in Table 4.5.

The following slice query can therefore extract each data item, as seen in Figure 4.51.

```
tag=keg slice uint16be([0:2]) as id int64be([2:10]) as sec
uint64be([10:18]) as nsec float32be([18:22]) as temp [22:] as name
| table
```



id	sec	nsec	temp	name
4608	63778652943	18269503	2.5	Keg 2
4608	63778652943	18024506	20.4370002747	System
4608	63778652943	17478514	4.06199979782	Keg 1
4608	63778652880	379962675	20.4370002747	System
4608	63778652880	379609680	4.06199979782	Keg 1
4608	63778652880	379163687	2.375	Keg 2
4608	63778652814	860138271	4	Keg 1
4608	63778652814	859769276	2.25	Keg 2
4608	63778652814	858896289	20.4370002747	System
4608	63778652748	298288691	3.93700003624	Keg 1

Figure 4.51: Sliced binary data

From the manual query it is possible to derive the following auto-extraction configuration:

```
[[extraction]]
  tag="keg"
  name="kegdata"
  desc="binary temperature control extractions"
  module="slice"
  params="uint16be([0:2]) as id int64be([2:10]) as sec
uint64be([10:18]) as nsec float32be([18:22]) as temp [22:] as name"
```

The complicated slice query now becomes:

```
tag=keg ax | table
```

Using filtering in the ax module and some math modules it is now possible to generate a graph showing the maximum temperature for each of the probes, as shown in Figure 4.52.

```
tag=keg ax id==0x1200 temp name | max temp by name | chart max by name
```

Additional filtering can be used to select only the keg temperatures and examine the temperature variance to see how well the control system is maintaining a constant temperature, as in Figure 4.53.

```
tag=keg ax id==0x1200 temp name~Keg | stddev temp by name | chart stddev by name
```

Using the auto-extractor and some basic math it is possible to dissect the binary data and clearly see a periodic engagement of the compressor, which causes an oscillation of temperature over time.

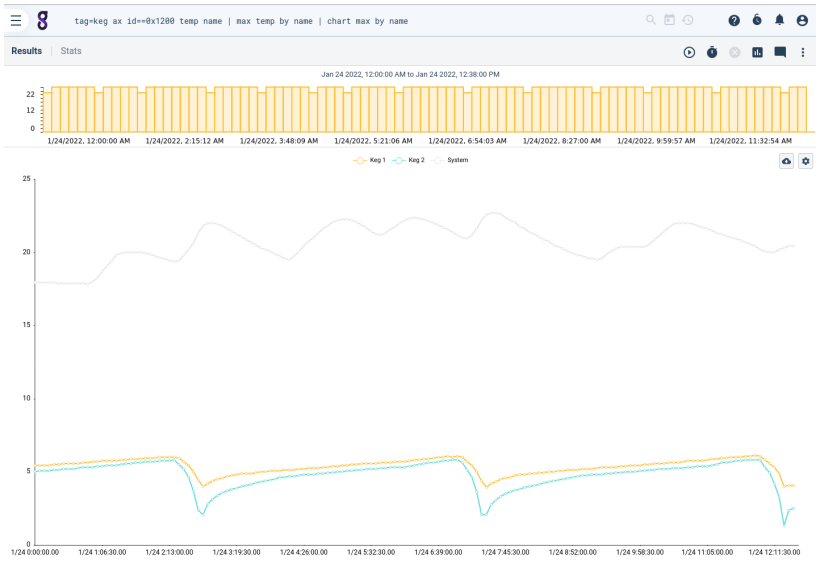


Figure 4.52: Keg temperature graph

Figure 4.53: Keg standard deviation graph

4.12.5 Hands-On Lab: Extractors

In this lab, we will show how to create an extractor to help work with CSV data. First, launch a Gravwell webserver+indexer container:

```
docker run --rm --net gravnet -p 8080:80 -d --name gravwell gravwell:base
```

Log into the web GUI (http://localhost:8080) and leave the page open.

Next, we'll use the ingester container image to run the CSV generator tool. This will ingest 100 CSV-formatted entries under the tag "csv":

```
docker run --rm --net gravnet --name ingesters -it -e \
GRAVWELL_CLEARTEXT_TARGETS=gravwell:4023 gravwell:ingesters \
/opt/gravwell/bin/csvGenerator -clear-conns gravwell:4023 -tag-name csv
```

A quick search on the csv tag should show the raw entries, as in Figure 4.54.

```
tag=csv
```

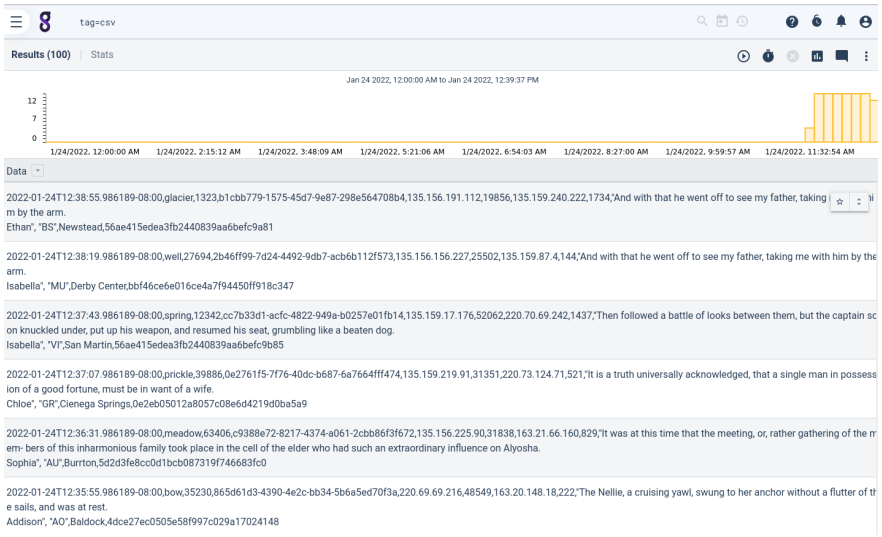


Figure 4.54: Raw CSV entries

We could manually pull out fields from these entries by specifying arguments to the csv search module, but a faster way is to create an extractor. First, we must gather the necessary components to build an extractor:

- tag: "csv"
- name: We will name this extractor "csv_extract"
- description: "CSV generator extraction"
- module: "csv"
- params: Here we must decide what to call each column of CSV entries. Based on our knowledge of the CSV generator, we know the columns are: a timestamp, an "application name", a random integer ID, a UUID, an IP address, a number which could be a TCP port, another IP, another port, a quote from a piece of literature, a country code, a city name, and a hex-encoded IP address. From this we will decide to name the columns "ts, app, id, uuid, ipA, portA, ipB, portB, msg, country, city, hexIP"
- args: none needed.

Open the menu in the Gravwell UI, then select the "Extractors" page. Click the "Add" button and populate it with the values above as shown in Figure 4.55

We can test it by running the query `tag=csv ax | table`, which will extract and display all fields. The results should resemble Figure 4.56.

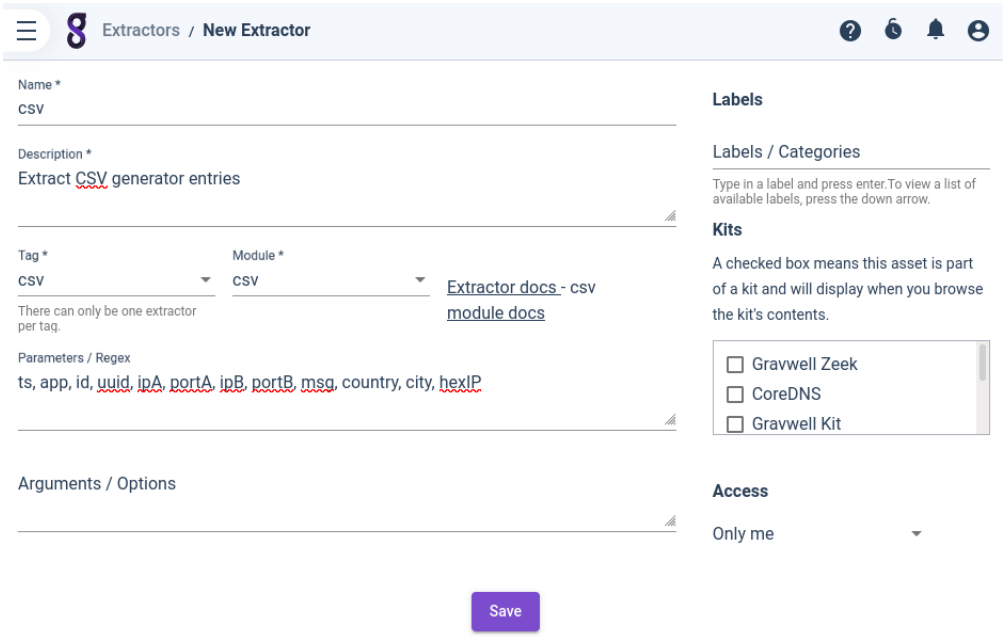


Figure 4.55: Creating a new extractor

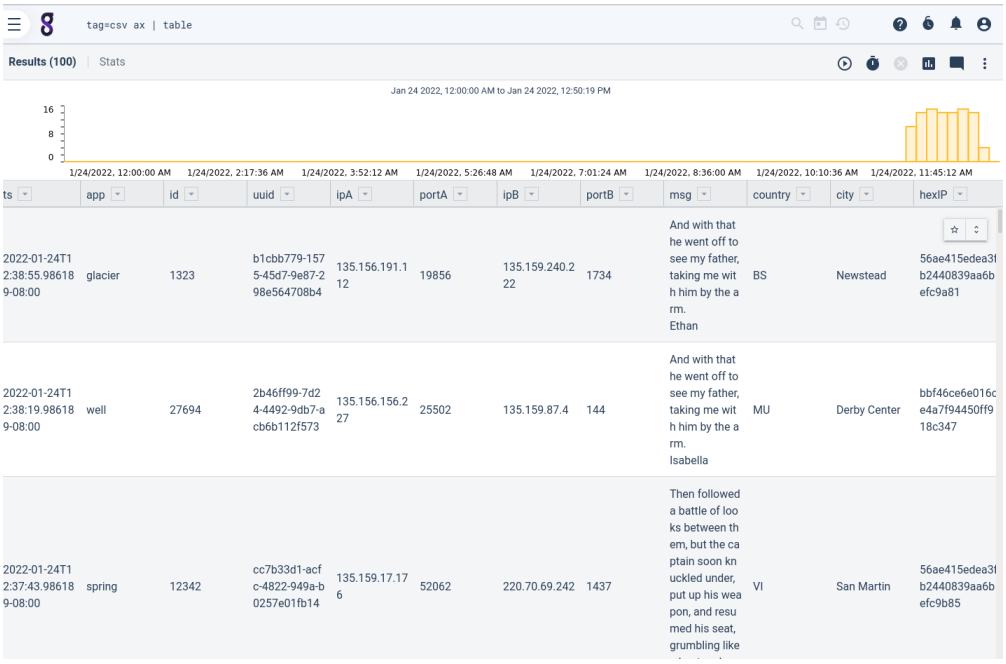


Figure 4.56: Testing new extraction

To clean up after the experiment, run:

```
docker kill $(docker ps -a -q)
```

4.13 Backgrounded and Saved Searches

Backgrounding a search allows a user to do other things while a search completes–it is conceptually similar to running a Unix command with an '&' at the end of the command line. Searches can be launched in the background by selecting the 'Background Search' button on the New Search page, as seen in Figure 4.57.

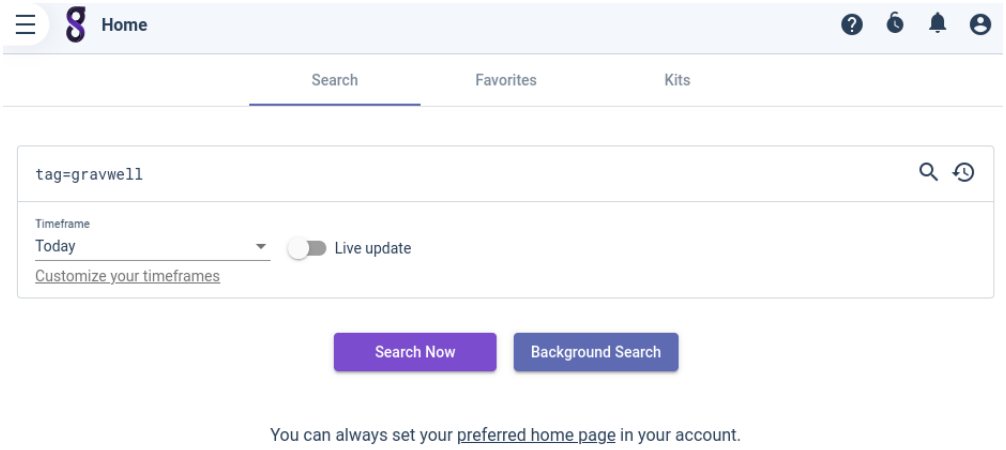


Figure 4.57: Starting a search in the background

Or a running search can be sent to the background from the menu, as in Figure 4.58.

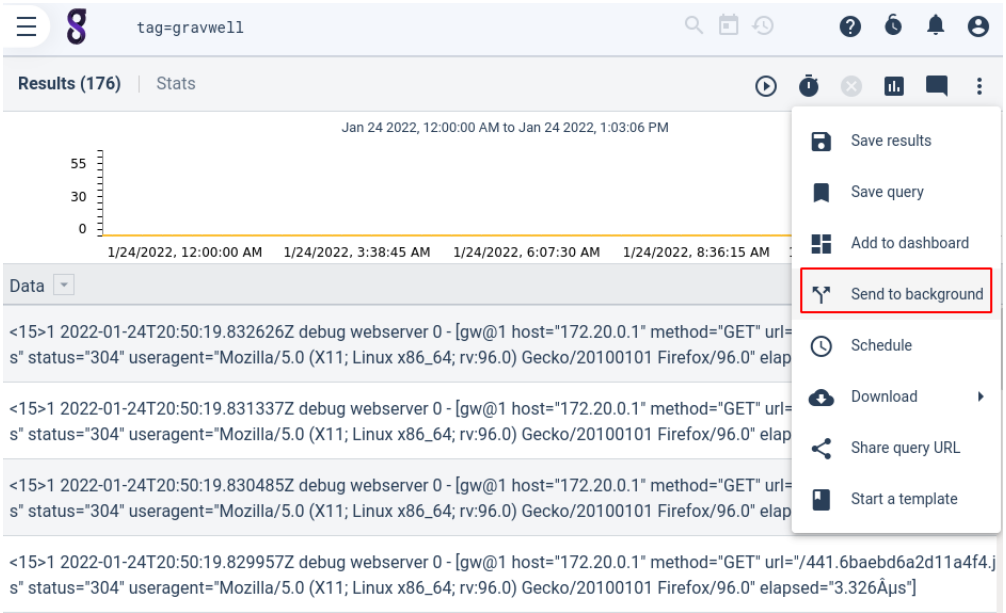


Figure 4.58: Backgrounding a running search

In either case, backgrounding a search frees the user to do other things while the search runs. The results of the search can later be viewed in the Persistent Searches page (Figure 4.59).

Note that a backgrounded search is not guaranteed to persist across server restarts. To keep the results of a search permanently, either select 'Save' in the search's menu on the Persistent Searches page (Figure 4.60) or select 'Save Results' from the menu on the search results page (Figure 4.61).

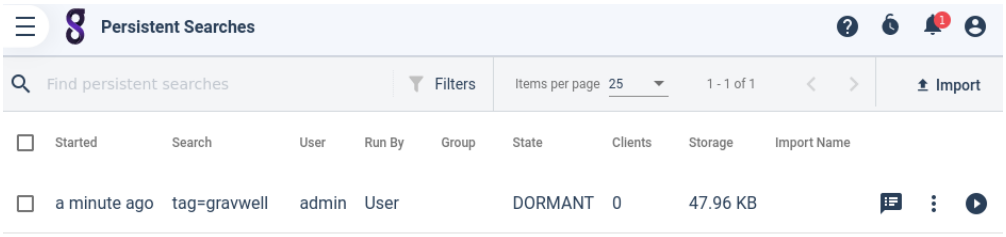


Figure 4.59: Persistent searches page

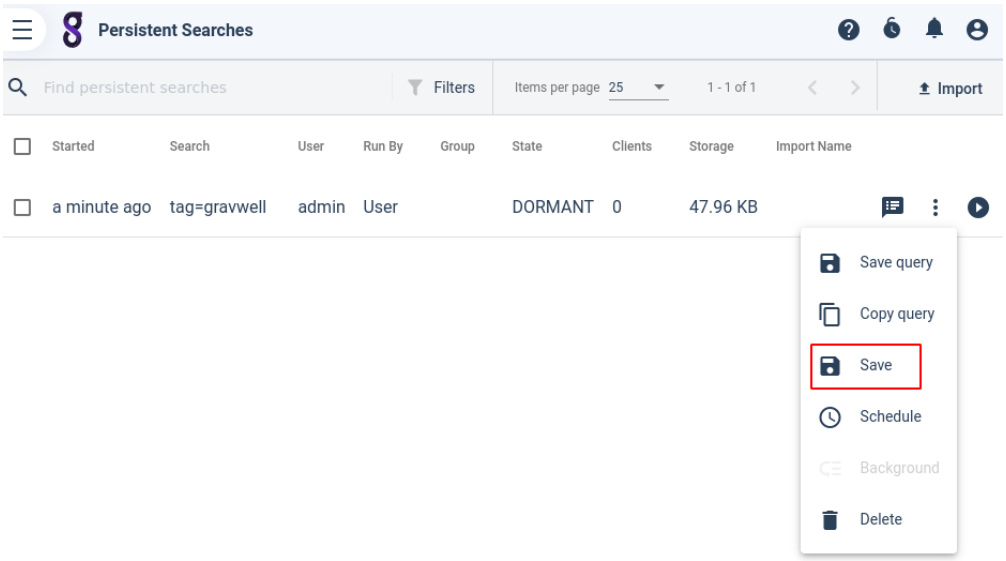


Figure 4.60: Saving a search from the persistent searches page

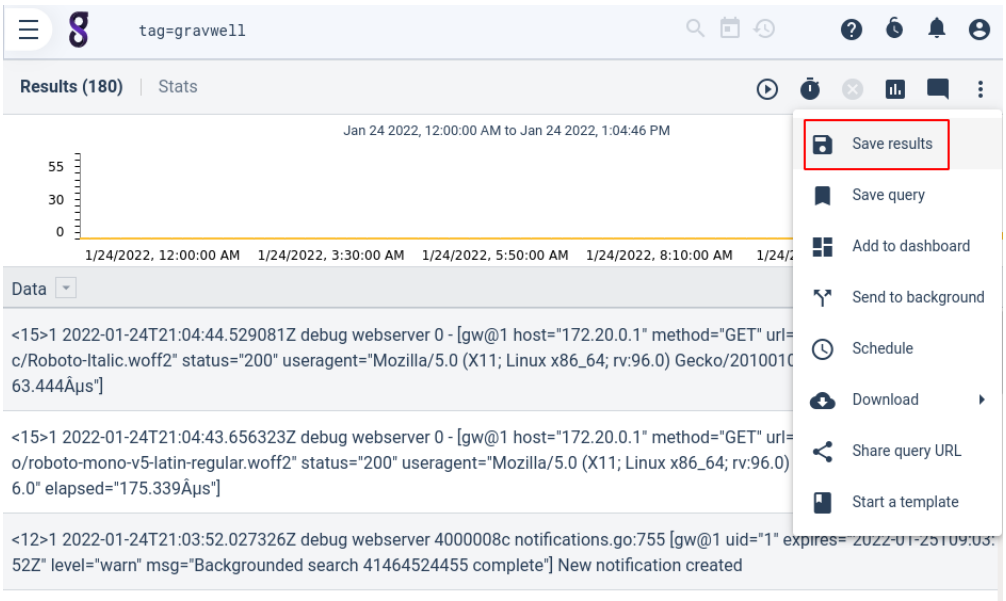


Figure 4.61: Saving a search from the search results page

Be aware that saved searches take up space on the disk, and the Gravwell administrator may choose to place restrictions on how much disk space users are allowed to consume for search storage. It is good practice to delete old saved searches when no longer needed!

4.14 Permissions, Groups, and Sharing Results

Any given user may belong to multiple groups, which are assigned by the administrator. Users can then choose to share their search results, dashboards, resources, and other things within the Gravwell system with members of a particular group. In general, if something can be shared with a group, the GUI will show a list of checkboxes, one per group. Checking the box shares the item with that group.

The administrator creates groups on the admin-only Groups administration page. In Figure 4.62, the administrator is creating a group named "foo" which contains the users John and Bob.

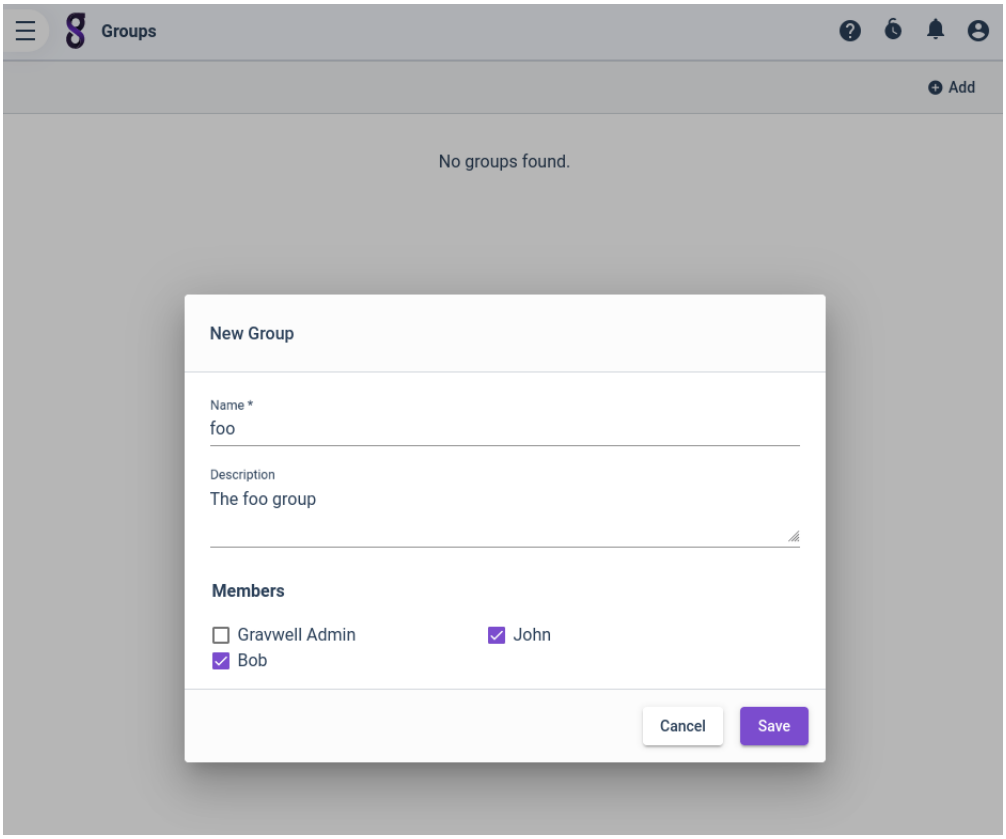


Figure 4.62: Creating a new group

Users can share the results of a search with a group from the Persistent Searches page by selecting the 'Edit group' button for that search, as shown in Figure 4.63.

Once shared with a group, all members of that group will see the search in their Persistent Searches listings. A user may optionally select a default group for all their searches; this means every search the user performs will by default be visible to members of that group. This can be set from the Preferences tab of the Account page (Figure 4.64).

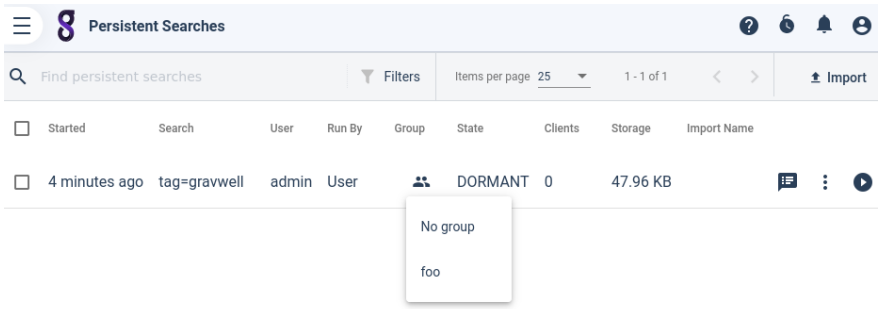


Figure 4.63: Sharing search results

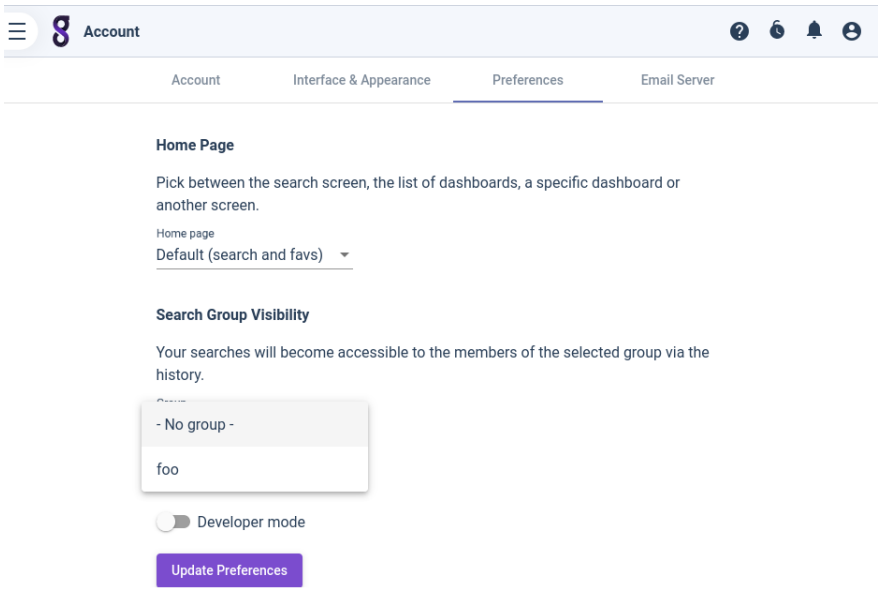


Figure 4.64: Setting default group

4.14.1 Hands-On Lab: Groups and Sharing

This lab will demonstrate user groups and search result sharing. First, launch a Gravwell webserver+indexer container:

```
docker run --rm --net gravnet -p 8080:80 -d --name gravwell gravwell:base
```

Next, we'll use the ingesters container to feed some entries into the indexer using the JSON generator:

```
docker run --net gravnet --rm -i --name ingesters gravwell:ingesters \
/opt/gravwell/bin/jsonGenerator -clear-conns gravwell:4023
```

Now, log in as the administrator (http://localhost:8080), open the Users page, and add two new users named 'john' and 'bob'. Use any passwords you like. Once you've created the new users, you should see three users in the User page, as in Figure 4.65.

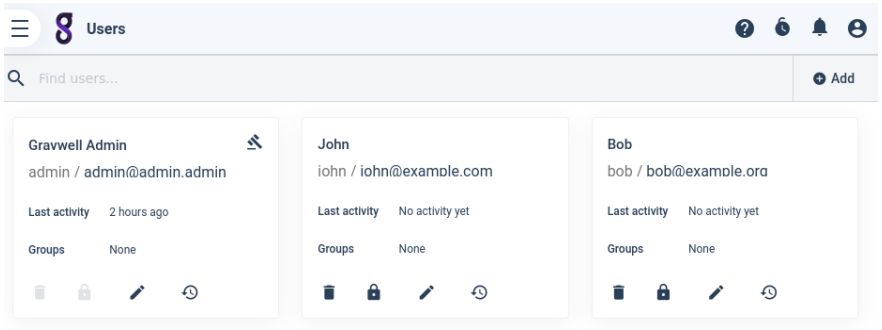


Figure 4.65: Additional user accounts

Then open the Groups page and add a new group called 'Test' containing those users, as shown in Figure 4.66.

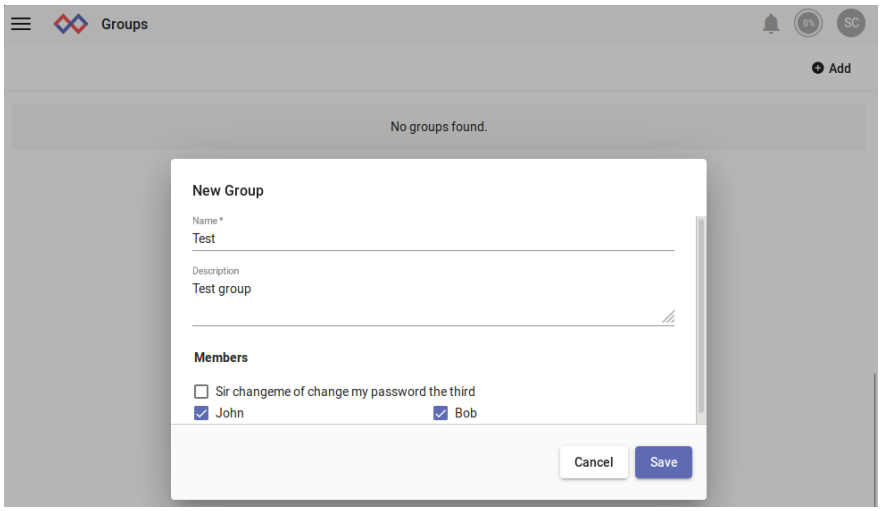


Figure 4.66: Creating a new test group

With that complete, open a new Incognito/Private browser window and log in as 'john', then run the search `tag=json` over the last day. Save the search, then go to the Persistent Searches page and make the search accessible to the new Test group (Figure 4.67).

Close the Incognito window and open a new one, then log in as 'bob'. Go to the Persistent Searches page; you

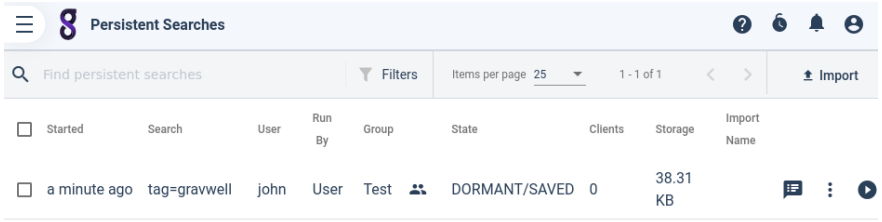


Figure 4.67: Saved search shared with Test group

should see the saved search owned by john. Open the saved search and verify that the contents are viewable by this user.

To clean up after the experiment, run:

```
docker kill $(docker ps -a -q)
```

4.15 Dashboards

Gravwell dashboards put relevant information in a heads-up format suitable for continuous monitoring and situational awareness. Dashboards are a collection of searches that are all executed in parallel when the dashboard is loaded. The results are placed into tiles which can be reordered or resized as desired.

Gravwell also supports "live" dashboards which automatically update the search data. Under the hood, this is done by re-launching the searches and swapping out the results when the new searches finish.

Dashboards are managed from the Dashboards page, as seen in Figure 4.68.

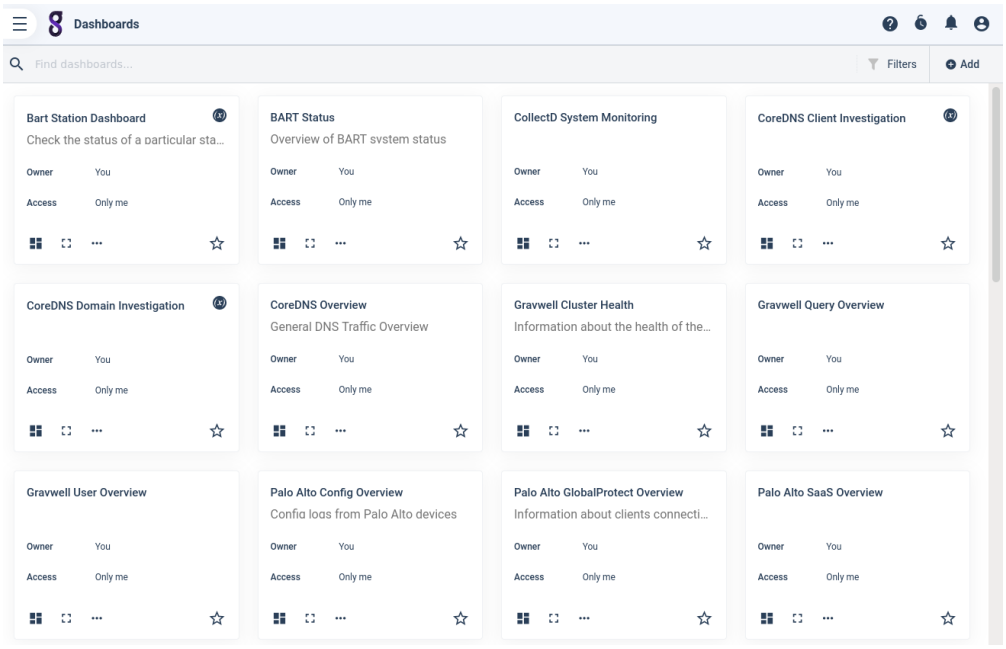


Figure 4.68: Dashboard management page.

New dashboards can be created by clicking the "Add" button in the upper right; this brings up the dialog box in Figure 4.69. The dashboard needs a name, a description, and a default timeframe. By default, all queries run within a dashboard use the same timeframe.

A newly-created dashboard is quite boring, as Figure 4.70 shows. There are two ways to add a new tile to a dashboard. Figure 4.71 shows how to add a tile from the query results page: open the 3-dot menu, click "Add to dashboard", then find the desired dashboard in the list, select it, and optionally specify a new name for the tile.

Figure 4.72 shows how to add a tile from within the dashboard page: click the "+" icon on the page, then fill out the resulting dialog. The "Query Settings" portion is where the query will be selected; clicking the pencil icon will open a sub-dialog with many different query sources, as seen in Figure 4.73. Queries may be typed in directly, selected from the query library, and so on. In Figure 4.72, we have chosen to enter a query manually and selected the line chart renderer.

Once a few tiles have been added to the dashboard, they can be rearranged and resized by clicking and dragging the tiles. Note that after making a change, you must click the "Save changes" popup which appears in the lower right corner.

4.15.1 Live Update

Dashboards can be configured to *live update*, meaning they will re-run queries and display new results after a set period of time. To enable this, click the 3-dot menu on the dashboard and select "Settings". Within the

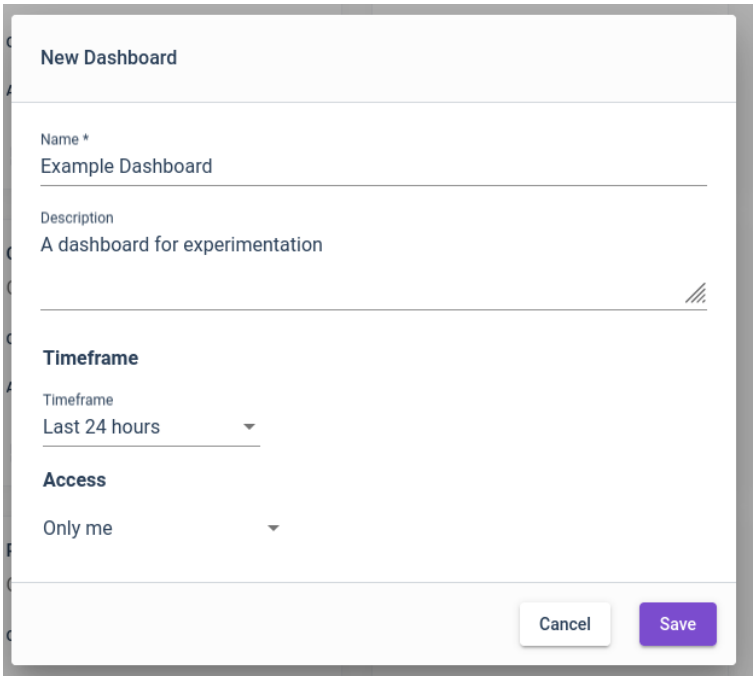


Figure 4.69: Creating a new dashboard.

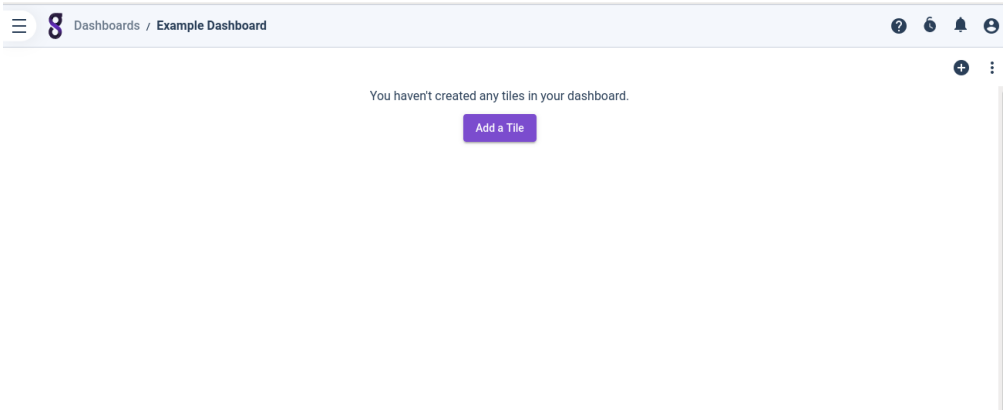


Figure 4.70: An empty dashboard.

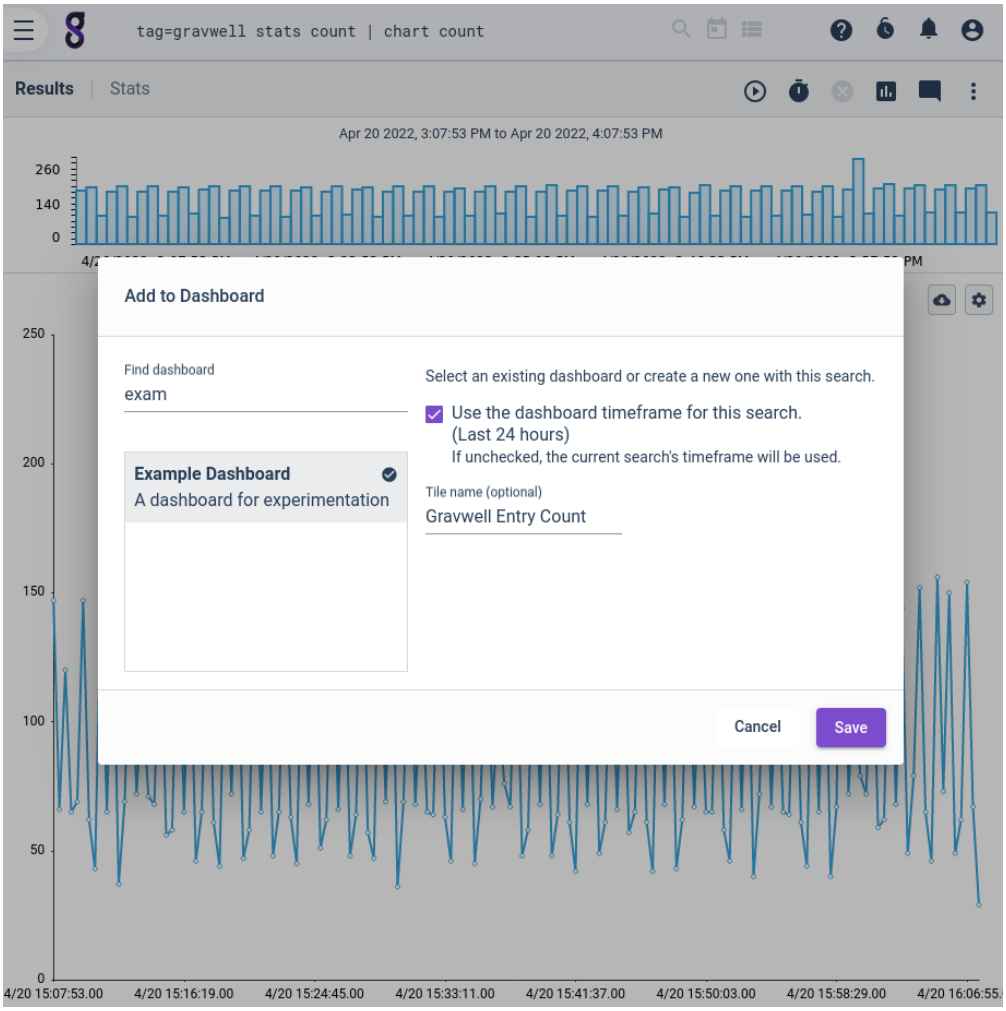


Figure 4.71: Adding a search to a dashboard

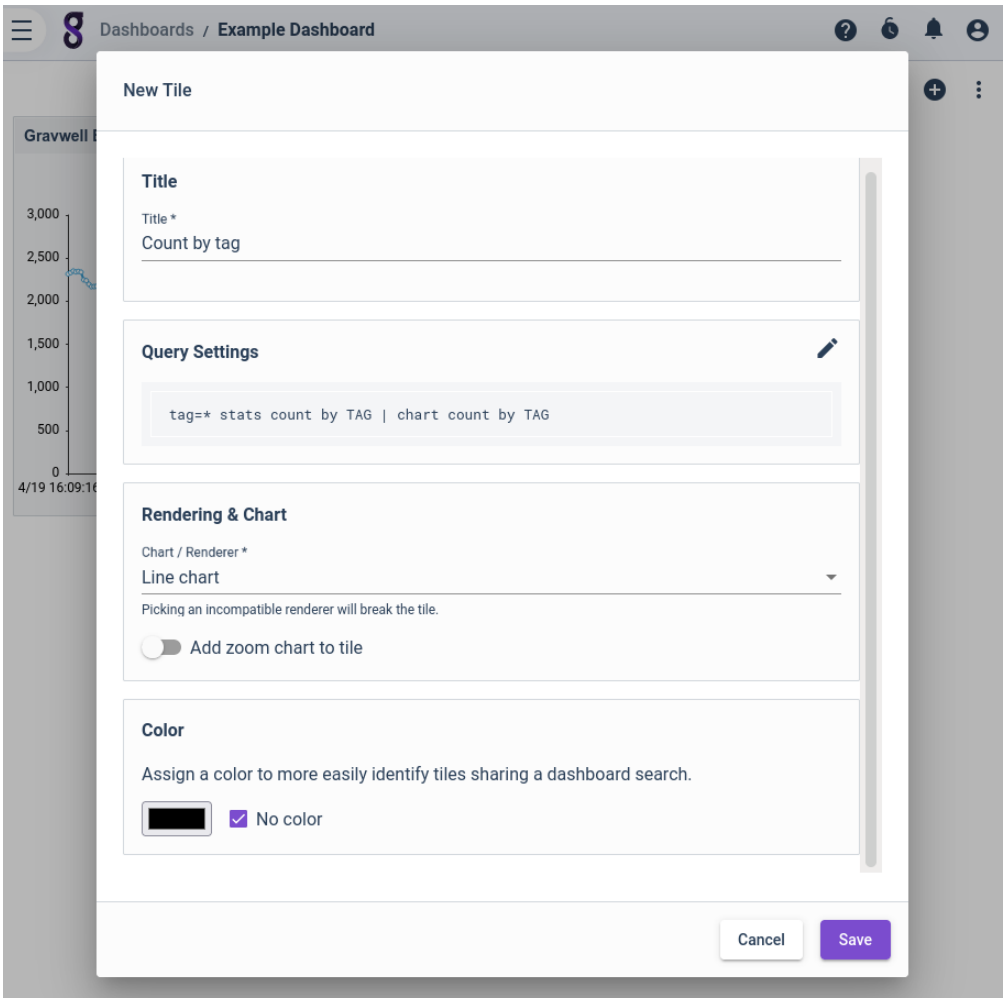


Figure 4.72: Adding a tile to a dashboard

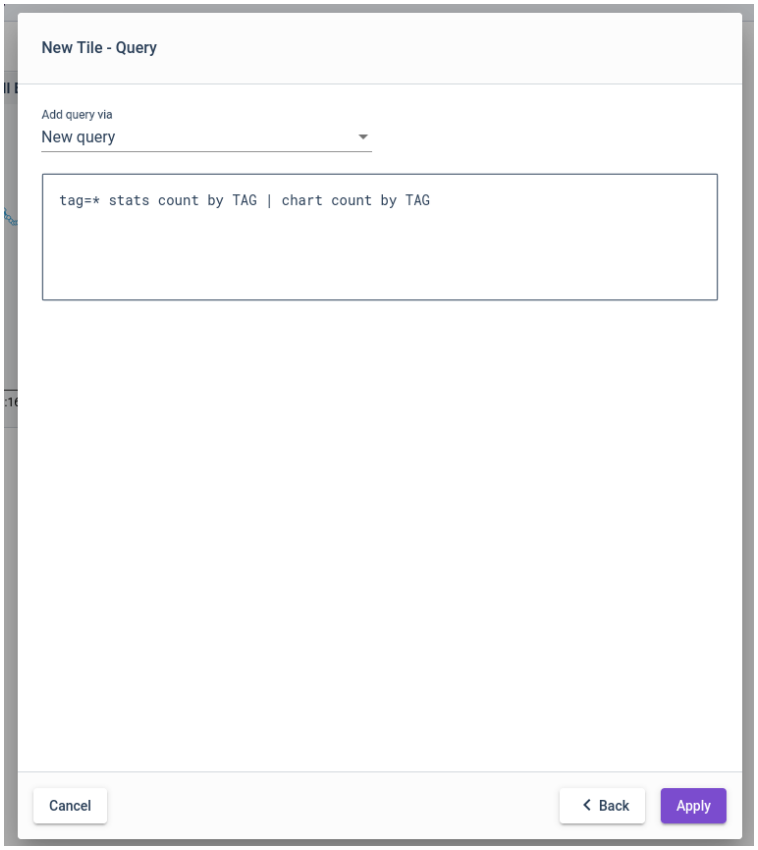


Figure 4.73: Query picker.

settings page, pick the “Timeframe & Live Update” tab, then turn the “Enable live update” toggle on, as seen in Figure 4.74. The update interval is configurable; if the queries in the dashboard cover a long timeframe or process a lot of data, consider setting the interval to a higher value so as to reduce the load on the system.

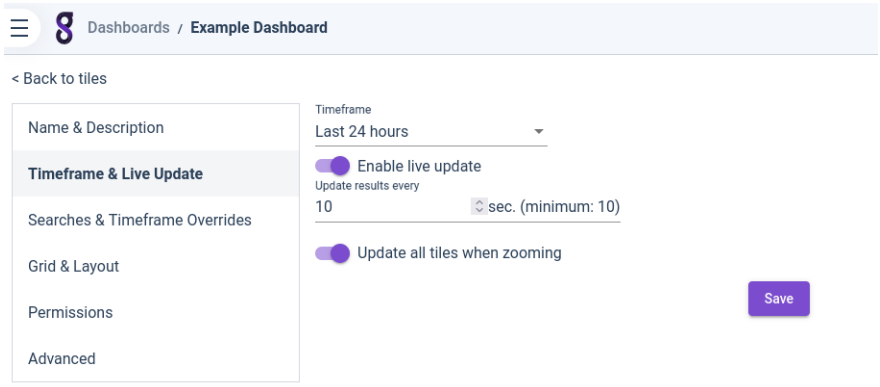


Figure 4.74: Live update settings.

4.15.2 Hands-on Lab: Network Activity Dashboard

For this exercise, we will be generating some Netflow data and creating a dashboard to provide some awareness of network activity. Let's start our docker containers for Gravwell, the netflow ingester, and a netflow generator.

```
docker run --rm --net gravnet -p 8080:80 -d --name gravwell gravwell:base

docker run --rm --net gravnet --name ingesters -it -e \
GRAVWELL_CLEARTEXT_TARGETS=gravwell:4023 gravwell:ingesters \
/opt/gravwell/bin/gravwell_netflow_capture

docker run -it --net gravnet --rm \
networkstatic/nflow-generator -t ingesters -p 2055
```

With Netflow data flowing in we can start to get our queries ready for addition to a dashboard. The goal of this dashboard is to break down network communications in some nice visuals that help us understand, at a glance, what the most common hosts are, what traffic is being utilized, where that traffic is going geographically, and potentially other data.

Let's start by running a search to show a chart of traffic by port so we can see which services are most used on our network:

```
tag=netflow netflow Port Bytes | stats sum(Bytes) by Port | chart sum by Port
```

Now, use the menu button in the upper right of the search results page to select "Add to dashboard". We have the option to create a fresh dashboard right from here, so let's do so (Figure 4.75).

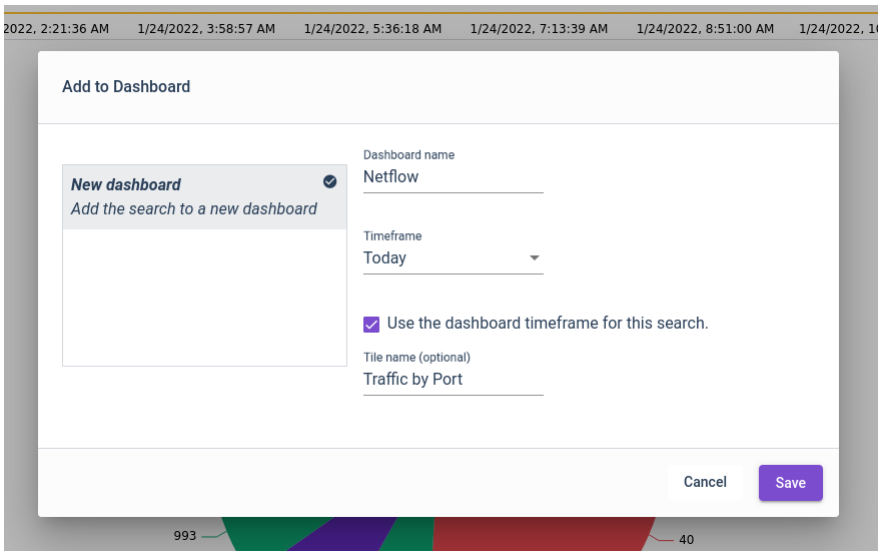


Figure 4.75: Adding search to a new dashboard

Click "View Dashboard" from the resulting popup message after you save. You should see the newly-created dashboard with the chart sitting nicely in a tile. Let's add one more search. Use the main menu to return to the new search page. This time, let's get a table together of which hosts are communicating the most:

```
tag=netflow netflow IP Bytes | stats sum(Bytes) by IP | table IP sum
```

Follow the same procedure to add this search to the dashboard but instead of selecting "New Dashboard" you can choose the already existing netflow dashboard. Give your tile a name like "Top Talkers", click save, and then let's move back to the dashboard view.

It would be nice to let the user zoom in on a specific time area of the dashboard results. This is accomplished by adding another tile as an "Overview" tile. The Overview renderer shows a chart of all log activity and allows the user to click-drag and zoom in on specific regions for more detail. To add an Overview tile, open the dashboard, then click the new tile button in the upper right. Give it a name ("Overview" works well) and set the query to any one of the existing queries in the dashboard. Then chose "Overview" from the renderer dropdown.

Dashboards can optionally sync up zooming of searches so that a zoom on one Overview renderer will cause all other tiles to zoom to the same range. This is controlled by the "Update all tiles when zooming" toggle option under the Settings page for the dashboard, in the "Timeframe & Live Update" section (see Figure 4.76). Open settings, enable that option, save, and then return to the dashboard view. Now try zooming in on a small region of the overview chart to see the other tiles respond.

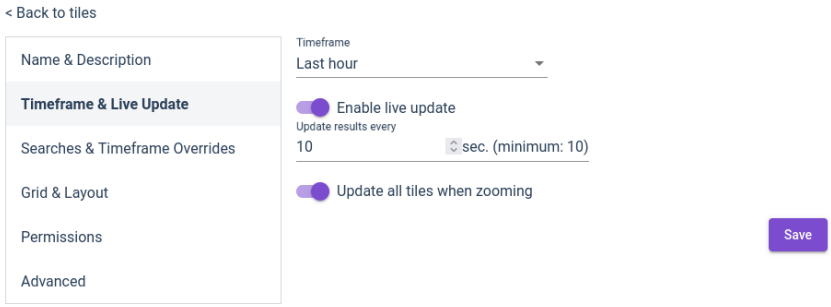


Figure 4.76: Live update and zoom sync options.

We should now have two tiles present that are showing us some information about our network communication, as well as an 'overview' tile which we can use for time exploration. As a final task, change the dashboard to use live updates. This will periodically refresh the underlying search data to keep the latest information available without having to refresh the page or manually re-run searches. Navigate to the dashboard's Settings page and turn on the "Enable live update" toggle in the "Timeframe & Live Update" section (see Figure 4.76). The default update interval of 10s is fine. Now, if you watch the dashboard, you should see it automatically refresh every 10 seconds.

To clean up after the experiment, run:

```
docker kill $(docker ps -a -q)
```

4.16 Templates

Templates are stored Gravwell queries which require one or more variables to run. This lets you build advanced queries which can investigate a particular IP address. They are particularly effective when inserted into a dashboard (section 4.15), or when coupled with an actionable (4.17).

Templates are managed via the templates page, accessed through the main menu under the 'Tools and Resources' section. A template consists of a name, a description, and the query itself. Inside the query, use words wrapped inside doubled percent signs to denote variables, e.g. `%%IPADDR%%` as seen in Figure 4.77.

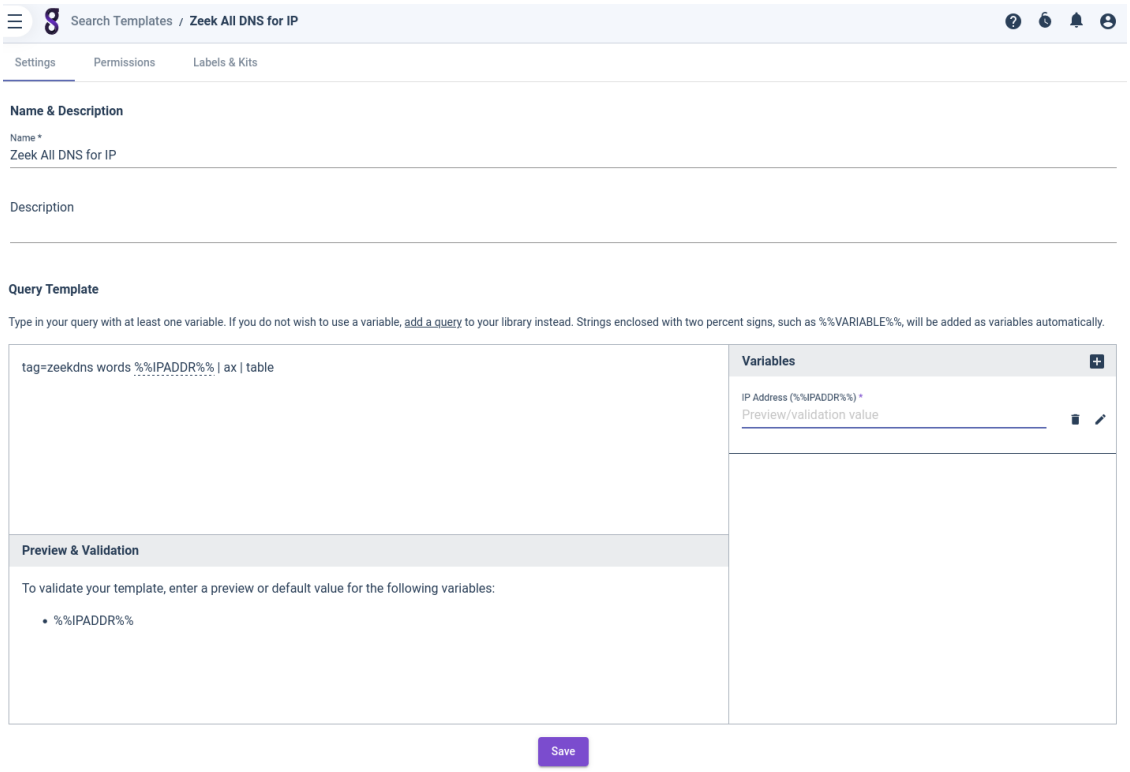


Figure 4.77: Editing a template

Once defined, a template can be run directly from the templates page by clicking the search button. A dialog (Figure 4.78) will open prompting the user to fill in the variable before launching the search. More often, though, a template is executed either by launching an actionable from search results, or in a dashboard.

Templates can be included in dashboards by selecting "Templates" in the "Add query via" dropdown when creating a new tile, as seen in Figure 4.79. When the dashboard is opened, the user is prompted for the variable values. Dashboards containing templates may also be launched from actionables (section 4.17).

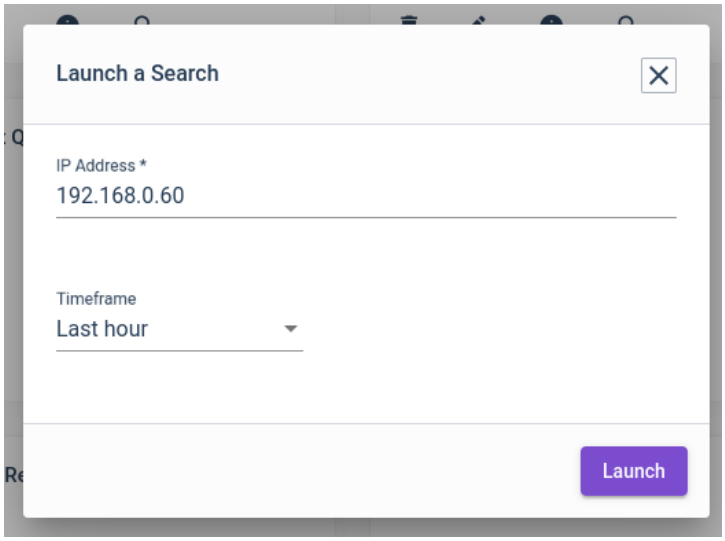


Figure 4.78: Launching a template

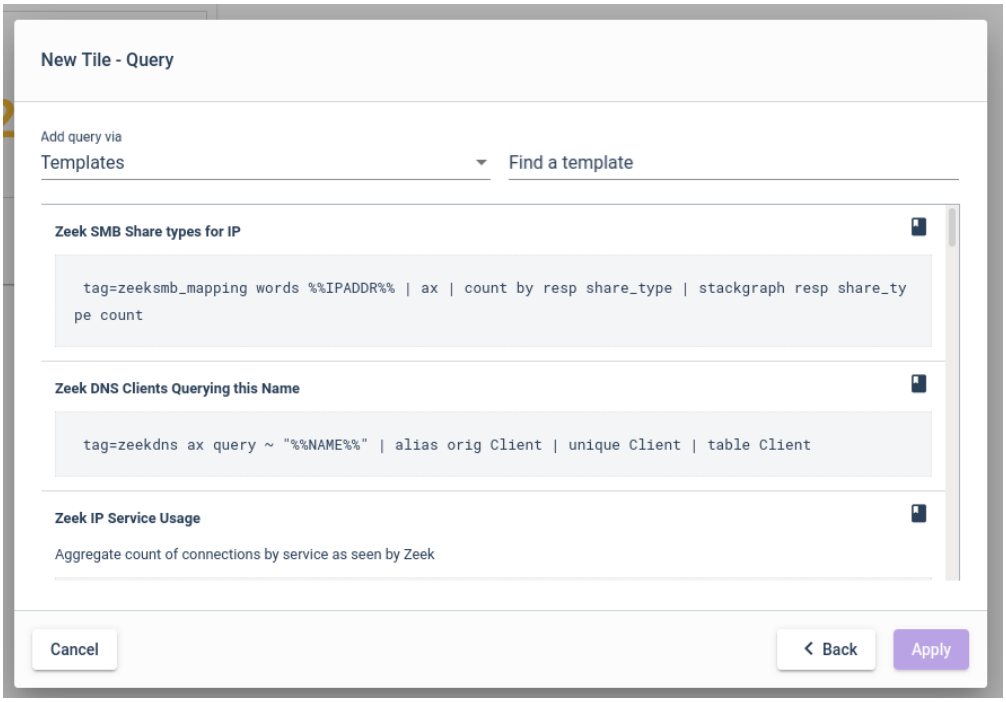


Figure 4.79: Adding a template to a dashboard tile

4.17 Actionables

Actionables provide a way to create custom triggers and menus that key on any text rendered in a query and take one or more actions when selected. Similar to an HTML hyperlink, actionables can be used to open external URLs that key on data, but actionables can also be leveraged to submit new Gravwell queries, launch dashboards, and execute templates.

Actionables are created by specifying one or more regular expressions, along with one or more actions. Gravwell automatically parses all text rendered with the table and chart renderers, bringing up appropriate actionable context menues when the text is clicked, as seen in Figure 4.80.

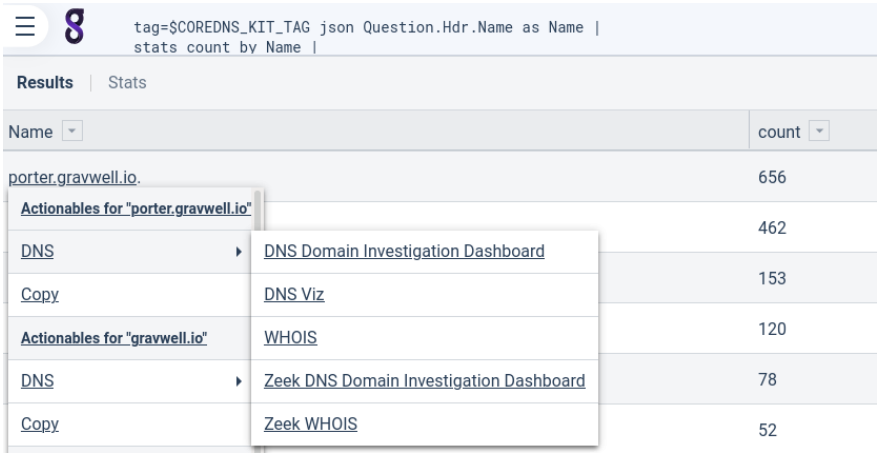


Figure 4.80: Actionables context menu

Actionables are made up of two components - triggers, which are simply regular expressions that Gravwell uses to match on text, and actions, which are the actions that can be taken on a matched trigger. An actionable can contain multiple triggers and/or multiple actions.

To get started with actionables, first open the Actionables menu, found in the main menu. Actionables are listed by name, and it's possible for two actionables to have the same name. By allowing actionables to have the same name, Gravwell can automatically group like actionables from different sources. For example, both the Netflow and CoreDNS kits provide actionables for IP addresses, and both are named "IP Address", as shown in Figure 4.81.

To create an actionable, you must define *actions* and, optionally, one or more *triggers*.

4.17.1 Actions

Actions provide operations that can be executed on text. An actionable can contain any number of actions. Actions include opening URLs, launching other searches, and more.

The _VALUE_ Variable

Some actions use the text of a capture group from the trigger to be used in the action itself. For example, we can use the trigger regex to extract a particular word in a string:

```
The color is (?<color>.*)
```

The capture group contents can then be used in a URL, using `_VALUE_` for the matched text, e.g. `https://en.wikipedia.org/wiki/_VALUE_`

The keyword `_VALUE_` is the default placeholder for the matched text, and can be changed in some actions.

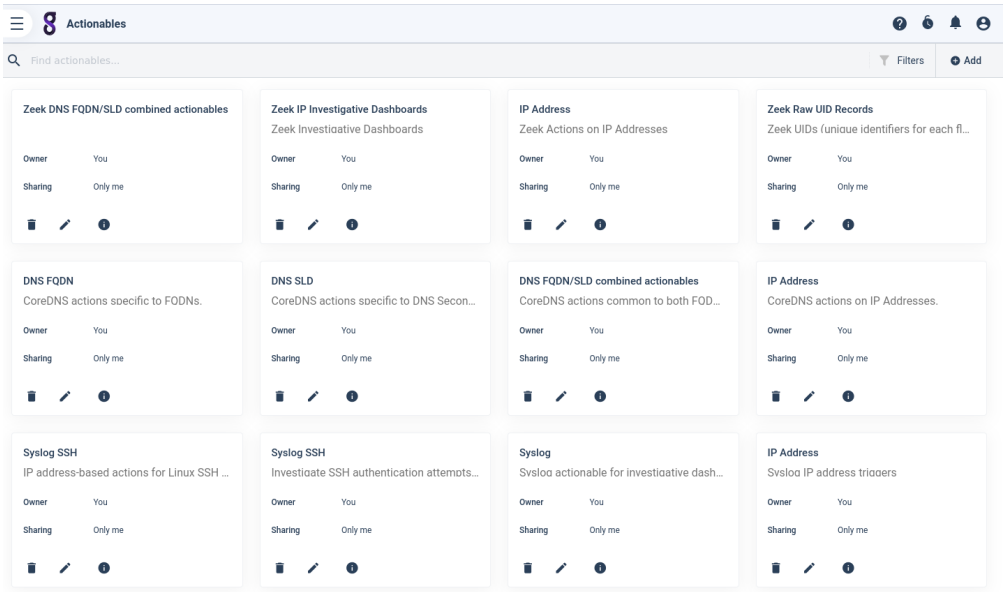


Figure 4.81: Actionables duplicate names

Action Types

Gravwell provides several actions, and an actionable can use any or all actions in a single actionable.

- Run a Query: This action runs a new query, replacing the string `_VALUE_` with whatever text was matched by the actionable.
- Execute a Template: The template action runs a pre-made template, using the matched text as the input variable to the template.
- Launch a Dashboard: The launch dashboard action opens a dashboard. If the dashboard has template variables, the user is prompted to select which variable to populate with the matched text.
- Open a URL: The URL action supports opening a new window/tab with a given URL and matched text, and additionally provides a set of timestamp options for providing the time range arguments from the search that the actionable triggered on. The "Open in a modal" option opens the URL in a window within the current Gravwell instance, similar to an HTML iframe.
- Run a Saved Query: Run a query from the Query Library.

Triggers

A trigger is a JavaScript regular expression that determines if an actionable should be displayed for any given piece of text. For instance, one might use `(?:[0-9]{1,3}\.){3}[0-9]{1,3}` as the trigger to match an IPv4 address. A trigger can be configured to highlight all matching text in query results with an underlined hyperlink which opens the actionables menu when clicked ("Click + text selection"), or it can be configured so that the actionable menu only pops up when the user explicitly selects text which matches the regular expression ("Text selection"). Figure 4.82 shows an example of a trigger on an actionable which will make any IPv4 addresses clickable.

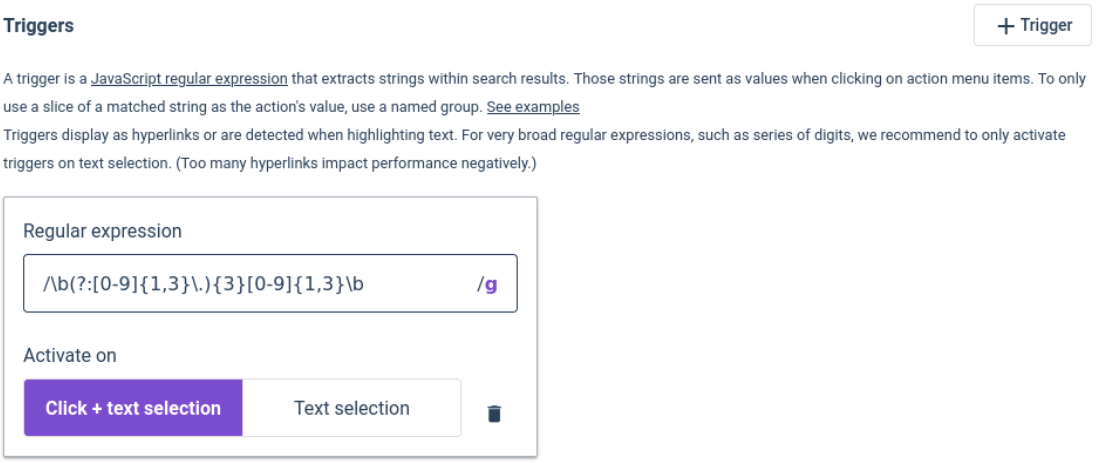


Figure 4.82: Example actionable trigger.

4.18 Compound Queries

Compound Queries is an extension to the query language that allows you to perform multiple, in-order, queries, and use the output from a previous query anywhere in the pipeline of the next, similar to an SQL JOIN. You can combine multiple queries together as a single "compound" query in order to leverage multiple data sources, fuse data into another query, and simplify complex queries. Gravwell's compound query syntax is a simple sequence of in-order queries, with additional notation to create temporary resources that can be referenced in queries later in the sequence.

A compound query consists of a main query (the last query in a sequence), and one or more inner queries. The main query is written just like a normal query, while inner queries are always wrapped in the notation `@<identifier>{<query>}`. Queries are separated by a semicolon and whitespace is ignored.

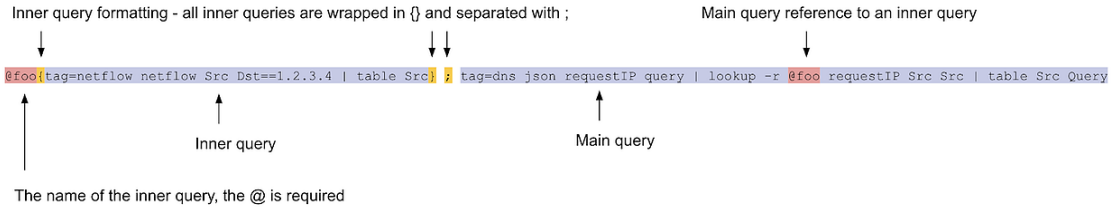


Figure 4.83: Parts of a compound query

For example, below is a compound query that has 2 inner queries and a main query.

```
@Q1{tag=default grep foo | json foo.bar foo.data};
@Q2{tag=syslog ax | table Event Payload};

tag=default ax
| lookup -r @Q1 match bar bar data
| lookup -r @Q2 Event Event Payload
```

Inner queries generate named resources in the form of `@<identifier>`. These can be used as regular resources with any module that supports table-based resources. Supported modules are shown in table 4.6. Unlike real resources however, named resources in a compound query are ephemeral and scoped - they exist only while the query is running and are visible only to compound query in which they were created.

Module dump	**Notes**
enrich	
ipexist	Inner queries must use the table module with the -format ipexist flag
iplookup	Inner queries must use the table module with the -format csv flag
lookup	
anko	Anko scripts can read from named resources

Table 4.6: Supported modules that can use compound query resources

Named resources are scoped to the compound query they exist in, and are ephemeral - they are only accessible to other queries in the compound query, and are deleted as soon as the query is completed.

For example, say we have both DNS query and IP-level connection data under the tags "dns" and "conns", and we want to filter connection data down to only connections that *didn't* first have a corresponding DNS

query; these are potentially suspicious IP addresses, since users typically access services via DNS names. We can use compound queries to enrich our first query with DNS data and filter.

Let's start with the inner query:

```
tag=dns json query answers | table query answers
```

This produces a table seen in Figure 4.84

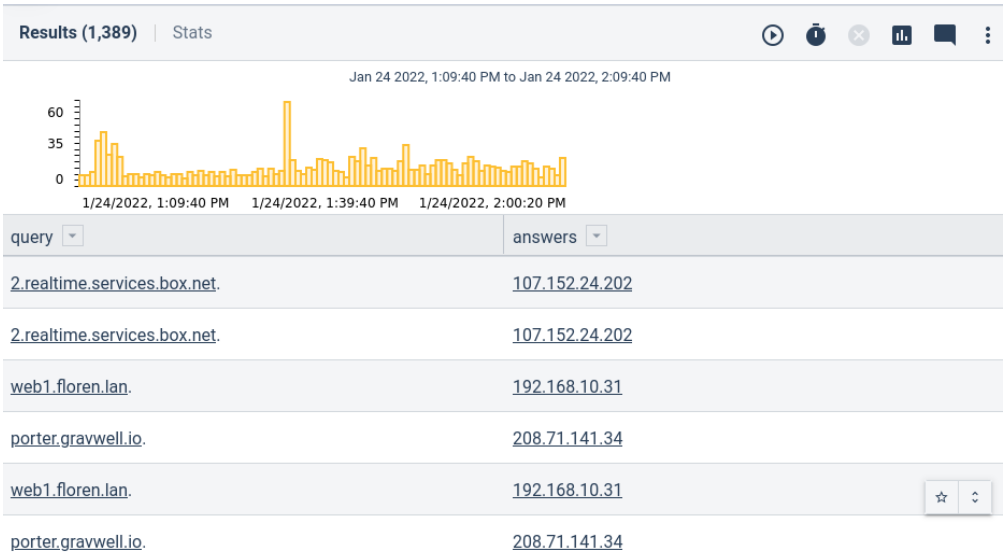


Figure 4.84: The beginning of an inner query

In the inner query, we simply create a table of all queries and answers in our DNS data. Since this is an inner query, we need to give it a name so later queries can reference its output, and wrap the query in braces. We'll call this inner query "dns":

```
@dns{tag=dns json query answers | table query answers}
```

In the main query, we use our connection data, and use the lookup module to read from our inner query "@dns":

```
tag=conns json SrcIP DstIP SrcIPBytes DstIPBytes
| lookup -s -v -r @dns SrcIP answers query
| table SrcIP DstIP SrcIPBytes DstIPBytes
```

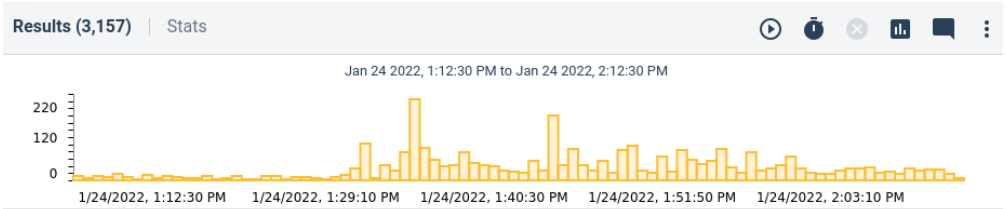
This query uses the lookup module drop (via the -s and -v flags) any entry in our conns data that has a SrcIP that matches a DNS answer. From there we simply create a table of our data.

We wrap this into a compound query simply by joining the queries together and separating them with a semicolon:

```
@dns{ tag=dns json query answers | table query answers };

tag=conns json SrcIP DstIP SrcIPBytes DstIPBytes
| lookup -s -v -r @dns SrcIP answers query
| table SrcIP DstIP SrcIPBytes DstIPBytes
```

This gives us a table (Figure 4.85 of just those connections that *didn't* have a corresponding DNS query.



SrcIP	DstIP	SrcIPBytes	DstIPBytes
192.168.0.111	129.159.201.182	2422	6931
192.168.0.111	3.65.178.143	0	0
192.168.0.111	192.168.0.254	39	113
192.168.0.111	129.159.201.182	2453	9523
192.168.0.111	129.159.201.182	2422	6540
192.168.0.30	8.8.8.8	64	108

Figure 4.85: Results from a complete compound query

Chapter 5

Indexers and Well Configuration

The two core components of a Gravwell deployment are the Indexer and Webserver. If either of those components are not properly configured, Gravwell will not function. The indexer is the most important component, as it is responsible for actually storing, maintaining, and querying data. In this section we will examine the role of a Gravwell indexer and how to appropriately configure and debug it.

Gravwell follows the principle of least privilege: the shell and Debian installers will create an unprivileged user and group named `gravwell`. Gravwell files and resources are stored in `/opt/gravwell` which is owned by the `gravwell` user. Most components have no special permissions and cannot affect sensitive system files. The exceptions are the webserver, simple relay ingester, file follower ingester, and PCAP ingester. The webserver and simple relay ingesters are given the `CAP_NET_BIND_SERVICE`[1] capability which allows them to bind to low ports; this is so we can serve web pages on ports 80 and 443 and listen for syslog packets on ports 514 and 601. The file follower ingester has a default systemd unit file which starts the process under the user `gravwell` and group `adm`,which allows it to access typical file based system logs in `/var/log`. The PCAP ingester is given the `CAP_NET_RAW` capability so that it can bind to raw sockets[2]. The special groups or capabilities can be removed, but the webserver will not be able to bind to low ports and the file follower ingester may not be able to read system logs.

5.1 Indexer Configuration

Gravwell always ships with a functional gravwell.conf that provides a basic deployment right out of the box. However, with very large or very complex deployments, tuning the configuration can yield much better performance and stability. Both indexers and webservers can share the same configuration file, which is located at `/opt/gravwell/etc/gravwell.conf` by default. The configuration file that is shipped in a typical Gravwell installer is very minimal and represents a basic configuration. The shell installer and Debian package both contain logic to inject randomly generated tokens into each required authentication parameter. The published Docker image on Docker Hub[3] contains environment variables preconfigured to set the authentication tokens. The preconfigured environment variables for the Docker deployment are not unique and not secure. If you plan to use Gravwell in Docker, you must override these configuration values using either Docker secrets or custom values.

[1] http://man7.org/linux/man-pages/man7/capabilities.7.html
[2] https://en.wikipedia.org/wiki/Network_socket#Raw_socket
[3] https://hub.docker.com/

5.1.1 Hands-on Lab: Misconfigured Indexer

For this hands on lab we will be deploying a slightly misconfigured Gravwell instance. We will examine error logs to identify the misconfiguration and correct it. To get started, let's cd to the working directory in our training folder named `Indexers/Lab-MisconfiguredIndexer`:

```
cd ~/gravwell_training/Indexers/Lab-MisconfiguredIndexer
```

Next, follow the steps to get your misconfigured Gravwell instance up and running:

1. Create a gravwell container using the `gravwell:base` image

   ```
   docker create --name test --net gravnet gravwell:base
   ```

2. Copy the `gravwell.conf` file into the container

   ```
   docker cp config/gravwell.conf test:/opt/gravwell/etc/
   ```

3. Start the container

   ```
   docker start test
   ```

4. Get the container IP address

   ```
   docker inspect --format \
   '{{.NetworkSettings.Networks.gravnet.IPAddress}}' test
   ```

5. Obtain a shell within the container

   ```
   docker exec -it test /bin/sh
   ```

Lab Questions

1. Are you able to view the Gravwell web portal?
2. Is the indexer process running?
3. What log files are available to identify the problem?
4. Why couldn't the indexer start?
5. What parameter in the gravwell.conf caused the failure?
6. What is the default value for the parameter and how would you change it?

Hands-on Lab Tips and Solutions

If a Gravwell instance is failing to respond, the first item of business is to identify whether or not the Gravwell processes are running. All Gravwell processes are prefixed with `gravwell_` so we can issue a `ps | grep gravwell` to get a list of running gravwell processes. In your container you should see the `gravwell_webserver` and `gravwell_searchagent` process, but *not* `gravwell_indexer`.

```
/ # ps | grep gravwell_
11 root    0:00 /opt/gravwell/bin/gravwell_webserver -stderr webserver
13 root    0:00 /opt/gravwell/bin/gravwell_searchagent -stderr searchagent
45 root    0:00 grep gravwell_
```

The indexer logs status information to a standard set of logs in `/opt/gravwell/log`, but if the process encounters a critical failure that prevents it from writing to the standard log files, it will write to a file in `/dev/shm/`. When a gravwell process fails and exits it will attempt to write the reason in `/opt/gravwell/log/error.log`, so check the error log to see if there is a message indicating the problem. Run `tail /opt/gravwell/log/error.log` and you should see something like:

```
03-14-2019 22:35:27.185 [8000020b] backend/main.go:561 Failed to create the search server:
listen tcp :4023:  bind:  address already in use
```

This message indicates that the Gravwell indexer process could not bind to port 4023 because something else was already bound to that port. The search server is the component within the Gravwell indexer that responds to commands from the webserver. The configuration parameter that controls which port it binds to is `Control-Port`. Open the gravwell.conf file you retrieved from the training server and examine its contents.

Let's change the `Control-Port` value to its default of 9404, copy the configuration file back into the container and restart it:

```
docker cp gravwell.conf test:/opt/gravwell/etc/
docker restart test
```

You should now be able to log into the gravwell web portal using the IP address discovered earlier. Do not `rm` the test container, as we are going to use it again for the next lab.

5.2 Well Configuration

Properly configuring a well can yield significant performance gains when querying data. Isolating data into different wells means that when you issue a query on a tag, the indexer can look at storage that is only associated with that tag. This allows for highly tuned configurations and storage arrangements. For example, let's assume that your Gravwell instance is consuming firewall logs, syslog, Netflow v5, and packet capture. The logs and Netflow may represent a few gigabytes of data per day and may have retention requirements, where the packet capture might produce hundreds or thousands of gigabytes per day and have no retention requirements. So while it might be very useful to consume packet data and hold it for a short time, it is not subject to the same rules or priority as the logs. Isolating it into its own well means that the high volume of packet data will have minimal impact on the logs and Netflow, and can be managed very differently.

To add a new well for an indexer we will need to add the well definition and associate the tag with it. Let's look at a well definition named "raw" that stores data in `/opt/gravwell/storage/raw` and is responsible for all entries with the tag `pcap` and `binary`:

```
[Storage-Well "raw"]
    Location=/opt/gravwell/storage/raw
    Tags=pcap
    Tags=binary
```

We can define as many wells as you would like; the only requirement is that every well must have its own storage location and none of the assigned tags may overlap. For example, the tag `pcap` cannot be assigned to multiple wells, and two different wells cannot use the same storage directory. If a well specifies a storage location that does not exist, it will attempt to create it. However, because the indexer is typically running as a restricted user it may not be able to. If you are specifying well storage locations outside of `/opt/gravwell/`, ensure that the location already exists and has the appropriate permissions. It should be owned by the user/group `gravwell/gravwell` and have RWX permissions enabled for the owner.

5.2.1 Hands-on Lab: Well Definitions

For this lab we are going to modify the gravwell.conf file to include a new well definition. We will use the existing container from the previous lab ("Misconfigured Indexer", 5.1.1)

Open the gravwell.conf file you fixed in the previous lab and append the following well definition:

```
[Storage-Well "syslog"]
    Location=/opt/gravwell/storage/syslog
    Tags=syslog
```

The entire gravwell.conf file should be:

```
[global]
### Web server HTTP/HTTPS settings
Web-Port=80
Insecure-Disable-HTTPS=true

### Other web server settings
Remote-Indexers=net:127.0.0.1:9404

### Ingester settings
Ingest-Port=4023
Control-Port=9404

### Other settings
Log-Level=INFO
Pipe-Ingest-Path=/opt/gravwell/comms/pipe

### Paths
[Default-Well]
        Location=/opt/gravwell/storage/default/

[Storage-Well "syslog"]
        Location=/opt/gravwell/storage/syslog
        Tags=syslog
```

Now copy it into the test container and restart the container:

```
docker cp gravwell.conf test:/opt/gravwell/etc
docker restart test
```

Obtain a shell within the docker container as shown below. Check that the indexer process has started and the well storage location was created. You should see some shards already in the default folder.

```
user@training:~/gravwell_training$ docker exec -it test /bin/sh
/ # ps -o pid,comm
PID   COMMAND
    1 manager
   11 gravwell_webser
   13 gravwell_search
   18 gravwell_indexe
   55 sh
   60 ps
/ # ls -l /opt/gravwell/storage
total 8
drwxr-x---    3 root     root          4096 Aug  5 17:32 default
drwxr-x---    3 root     root          4096 Aug  5 17:33 syslog
```

```
/ # ls -al /opt/gravwell/storage/default/
total 12
drwxr-x---    3 root     root          4096 Aug  5 17:32 .
drwxrwx---    1 gravwell gravwell      4096 Aug  5 17:33 ..
drwxr-xr-x    2 root     root          4096 Aug  5 17:32 76d3f
/ #
```

We can also validate that the new well was created within the GUI. Expand the left sidebar navigation menu and click "Storage, Indexers, & Wells" under "Systems & Health". From there, click on the indexer to see the two wells, default and syslog, as shown in Figure 5.1.

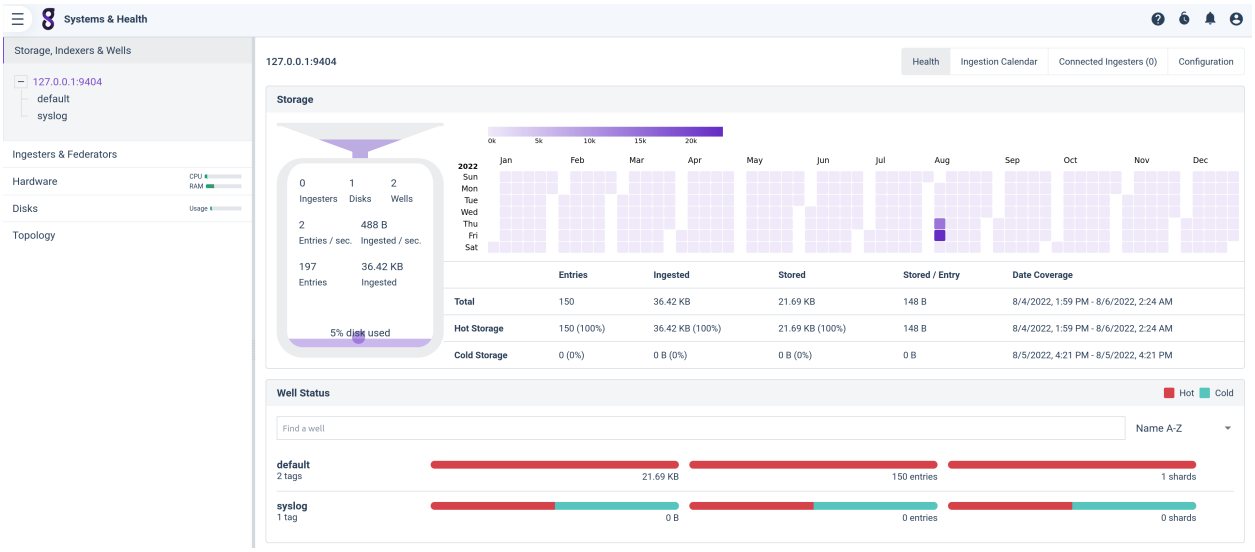


Figure 5.1: Wells and Indexers

Select the Configuration tab at the top to view configuration details for the wells as shown in Figure 5.2.

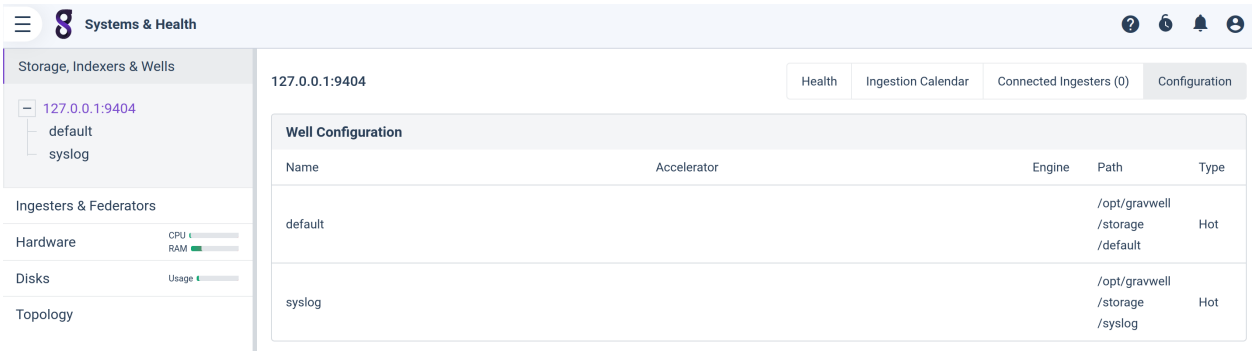


Figure 5.2: Wells Configuration Details

The indexer also automatically created the syslog tag when it found the new well configuration. Select "Tags" from the navigation menu and see that Gravwell now knows about the syslog tag, enabling you to search with it. There are no entries yet, as we haven't ingested anything, but the tag is there.

Do not `rm` the test container, as we are going to use it again for the next lab.

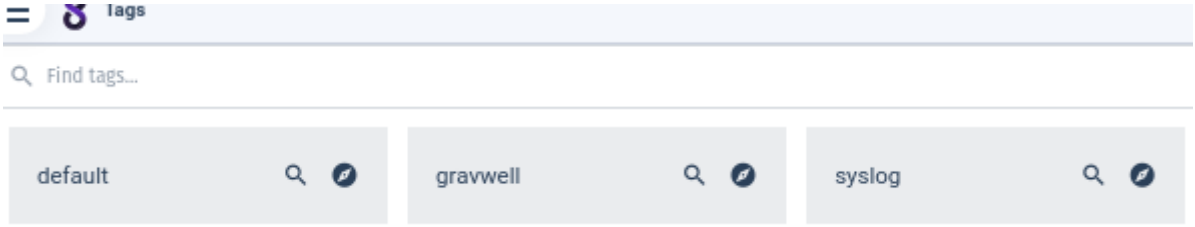


Figure 5.3: Tags

5.3 Well Ageout

Gravwell is designed to manage data sets with minimal user interaction; once a system is appropriately configured it will manage data sets and storage arrays on its own. Data ageout is one of the most critical configurations. Properly configured well ageouts prevent disk exhaustion, enable high speed working datasets, and allow for regulatory or retention compliance. This section will examine the available rulesets which control how Gravwell indexers manage stored data and how we can use multi-tiered storage to adhere to compliance requirements while still utilizing a smaller pool of high speed storage for day-to-day operations.

Data ageout can be controlled using three different constraints: time, total storage, and storage availability. Ageout is also configured on a per well basis, so each well can have entirely different ageout rules. The per well ageout rules enable Gravwell administrators to prioritize data and adhere to corporate or regulatory requirements without lumping every data source into the same rule set or priority. Data that must be captured but may not be useful in day to day operation can be stored directly on low-cost storage devices for long term storage. Data that may not have legal or corporate retention requirements but plays a central role in network management or security operations can use high speed storage and ageout to low-cost storage as needed. Gravwell ageout rules provide a tremendous amount of flexibility in managing how data is stored.

Gravwell indexers are very sensitive to data deletion. You *must* explicitly enable data deletion via the ageout controls using the `Delete-Cold-Data` and `Delete-Frozen-Data` parameters. If the respective parameters are not set to "true", an indexer will not delete data, even if the other configuration parameters indicate that it should. This additional configuration parameter is somewhat like a secondary sanity check, forcing you to fully acknowledge that Gravwell is allowed to delete data.

WARNING: Data ageout operates on entire shards and ageout constraints must be satisfied by an entire shard before an ageout executes. A typical shard size is 1.55 days, so if you set a time based ageout of 1 day, Gravwell won't perform an ageout until *all* data in a shard is more than 1 day old. This means that even though you specified an ageout duration of 1 day, the storage tier will hold up to 1.55 days. When an ageout occurs, the entire shard is aged out at once.

5.3.1 Time-based Ageout

Time-based ageout uses entry timestamps to determine their life cycle. Ageout is specified using time spans that define retention durations. Each storage tier can maintain an independent duration configuration. For example, we can define that entries are held in the hot storage pool until they are greater than 30 days old, at which point they are deleted or moved to a cold storage tier. Cold storage can then specify that data be held for 120 days before it is deleted or archived. Time-based retention is controlled by the `Hot-Duration` and `Cold-Duration` well parameters. Durations can be specified in days (d) or weeks (w). Because aging out data incurs some computation and storage costs, it may be beneficial to schedule the ageout for a time of day when users are not interacting with Gravwell, or when log data volumes are lower. The `Ageout-Time-Override` parameter allows you to set the time of day that the time-based ageouts occur–note that this time is specified in UTC!

Below is an example well configuration that stores data in the hot tier for seven days, then moves it to the cold tier where it is held for 16 weeks and then deleted. Ageout occurs at 1AM UTC.

```
[Default-Well]
        Location=/mnt/ssd/gravwell/default/
        Cold-Location=/mnt/hdd/gravwell/default
        Hot-Duration=7d
        Cold-Duration=16w
        Delete-Frozen-Data=true
        Ageout-Time-Override="1:00"
```

This is a well that does not maintain a cold storage tier; instead it simply deletes data once it is 90 days old:

```
[Default-Well]
        Location=/opt/gravwell/storage/default/
        Hot-Duration=90d
        Delete-Cold-Data=true
```

WARNING: The indexer host clock is critically important for accurate ageout controls. The timestamps on entries are entirely decoupled from the time on the host indexer. If you misconfigure the host clock on an indexer, ageout may execute prematurely, or not at all.

5.3.2 Storage-based Ageout

Storage-based ageout uses the total physical storage consumed to determine data lifecycles. Using storage-based ageout is convenient when managing smaller high-speed storage pools, managing specific storage usage, or when there are no retention requirements. Wells configure their storage-based ageouts using the `Max-Hot-Storage-GB` and `Max-Cold-Storage-GB` well parameters. Setting a storage constraint tells Gravwell that if a well storage tier goes over the specified storage limit, it should attempt an ageout immediately. Storage-based ageout rules operate entirely independently of time-based rules. If both time- and storage-based ageouts are configured and the storage tier exceeds the configured storage limit, the indexer will attempt an ageout regardless of what the time-based controls say.

This is a well configured with two storage tiers. The hot tier is configured to hold 10GB and the cold tier is configured to hold 100GB, so the entire well will hold 110GB of searchable data:

```
[Default-Well]
        Location=/mnt/ssd/gravwell/default/
        Cold-Location=/mnt/hdd/gravwell/default
        Max-Hot-Storage-GB=10
        Max-Cold-Storage-GB=100
        Delete-Frozen-Data=true
```

This is a well configured to use a storage constraint on the hot tier and a time constraint on the cold tier. This means it will keep 10GB in the hot storage device, but ensure that data is held for at least 90 days using the cold storage device:

```
[Default-Well]
        Location=/mnt/ssd/gravwell/default/
        Cold-Location=/mnt/hdd/gravwell/default
        Max-Hot-Storage-GB=10
        Cold-Duration=90d
        Delete-Frozen-Data=true
```

WARNING: Data ageout can take non-zero time. Ensure that you have ample storage space to continue ingesting new data while old data is moved between storage tiers. For example, if your hot storage tier has 128GB of available storage it is wise to configure the storage limit to 120GB so that you have 8GB of slack available while the ageout is in progress.

5.3.3 Storage Availability Ageout

Storage availability-based ageout a highly flexible system which enables indexers to use storage without hard constraints on time or stored data. The availability constraints are used to define a specific percentage of storage that the indexer *cannot* use, a sort of reserve space. The constraints are controlled by the `Hot-Storage-Reserve` and `Cold-Storage-Reserve` parameters. Storage reserves are calculated by each well and storage tier independently. This means that wells do not coordinate with each other and they do not treat disk usage by outside applications any differently than storage used by Gravwell.

This is a well definition which maintains a 10% reserve on the hot storage tier:

```
[Default-Well]
        Location=/opt/gravwell/storage/default/
        Hot-Storage-Reserve=10
        Delete-Cold-Data=true
```

This is a configuration with two wells, both using storage reserve-based constraints. Notice that both wells are based in `/opt/gravwell/storage/`; if we assume that storage locations are on the same storage device the ageout behavior may be unexpected. For example, if the storage device has 128GB of available storage, and the default well has shards that are 100GB in size while the syslog well has shards that are 1GB in size it will be extremely difficult to predict which well will actually delete its data. If the default well triggers first, the syslog well may be able to maintain months worth of data. If the syslog well triggers first it may delete all of its shards trying to hit the storage reserve.

```
[Default-Well]
        Location=/mnt/ssd/gravwell/default/
        Hot-Storage-Reserve=10
        Delete-Cold-Data=true

[Storage-Well "syslog"]
        Location=/opt/gravwell/storage/syslog
        Tags=syslog
        Hot-Storage-Reserve=20
        Delete-Cold-Data=true
```

WARNING: Using the availability constraints when a Gravwell indexer does not have exclusive control over an entire storage device can yield unexpected ageout behavior.

5.3.4 Hands-on Lab: Ageout

This hands-on lab will continue to use the Gravwell docker instance that we used in the previous labs. We are going to apply ageout constraints to the `default` and `syslog` wells and watch data move from the hot tier to the cold tier. First, let's make sure our previous container is running, then change to the `Lab-Ageout` directory in our training folder:

```
docker start test
cd ~/gravwell_training/Indexers/Lab-Ageout
```

Then we'll use the ingesters container (make sure you've loaded the ingesters container image as described in Section 2.5) to import the syslog data:

```
docker run -v $PWD/data:/tmp/data --rm -i --net gravnet \
gravwell:ingesters /opt/gravwell/bin/reimport -rebase-timestamp \
-clear-conns test:4023 -i /tmp/data/syslogdata -import-format json
```

In the command above, we spin up a new container and run the `reimport` ingester, telling it to read json-formatted entries from standard input.

Next, let's apply some ageout constraints to the two wells. We have provided a copy of the base gravwell.conf in the config/ subdirectory, so modify it and add the following constraints:

1. `Default well`
   (a) One storage tier
      i. Hot is at /opt/gravwell/storage/default
   (b) Hot tier holds 14 days of data
      i. Data is deleted after
2. Syslog well
   (a) Two storage tiers
      i. Hot location is /opt/gravwell/storage/syslog
      ii. Cold is at /opt/gravwell/cold_storage/syslog
   (b) Hot tier holds 7 days of data
   (c) Cold tier holds 1GB
      i. Data is deleted after

Once you have made your docker configuration changes copy the configuration file into the test container and start it.

```
docker cp config/gravwell.conf test:/opt/gravwell/etc/
docker restart test
```

Check the well configuration details in the GUI to verify that your indexer is properly configured. Selecting the Health tab for the indexer will show about 20k entries in the `syslog` well in hot storage.

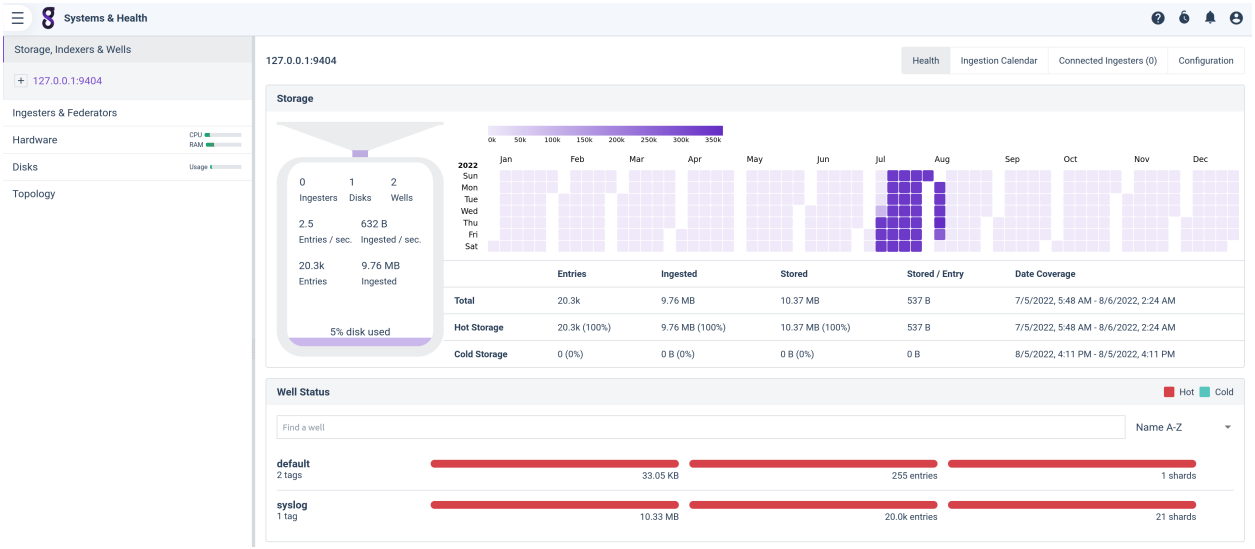


Figure 5.4: Data in Hot Store

The indexer will age the hot data into the cold well. Please wait to observe this change; it may take some time. You can also click on each individual well to see details for that one alone.

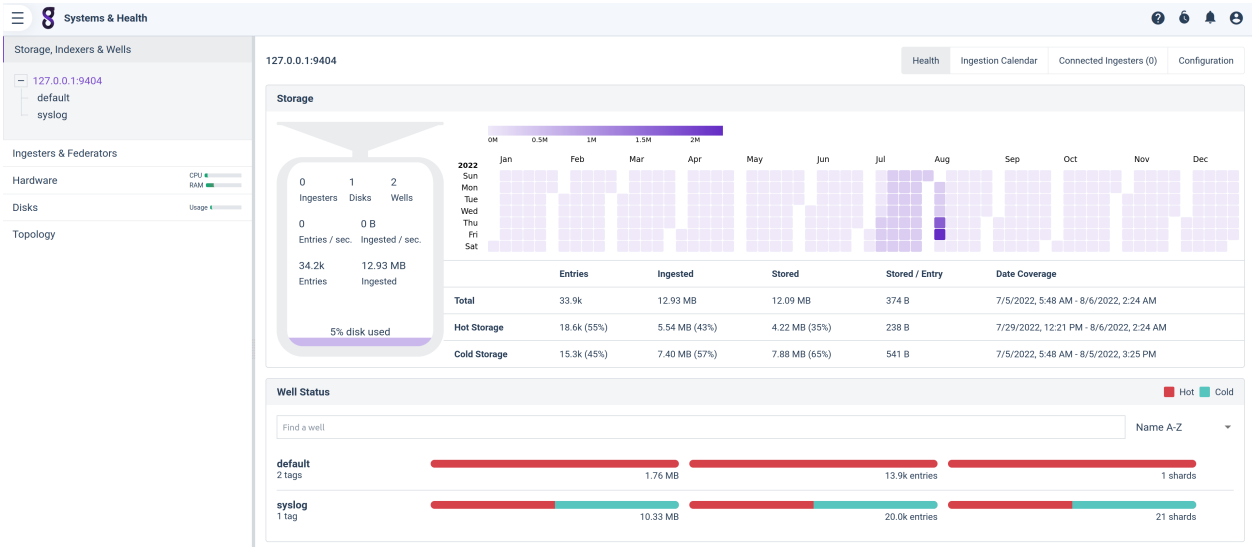


Figure 5.5: Data in Hot and Cold Stores

Your modified `gravwell.conf` well definitions should look something like this:

```
[Default-Well]
    Location=/opt/gravwell/storage/default/
    Hot-Duration=14d
    Delete-Cold-Data=true

[Storage-Well "syslog"]
    Location=/opt/gravwell/storage/syslog/
    Cold-Location=/opt/gravwell/cold_storage/syslog
    Tags=syslog
    Hot-Duration=7d
    Max-Cold-Storage-GB=1
    Delete-Frozen-Data=true
```

Do not `rm` the test container, as we are going to use it again for the next lab.

5.4 Replication

Hardware failures happen, drives crash, and humans mis-type commands; only a robust data backup solution can prevent catastrophic data loss. Gravwell implements a data replication system which enables transparent data backup. As data is ingested, Gravwell will automatically assign the data stream to a remote peer and copy the data shards to it. The replication system is extremely flexible, enabling tolerance to node, rack, and even data center failures. Indexers replicate data shards and tag mappings, which is sufficient to completely rebuild a node in case of total storage failure. Indexer configurations are not replicated, so maintaining an up-to-date `gravwell.conf` file is important.

This section will explore the two data replication mechanisms supported by Gravwell and examine the pros and cons of each. We will describe how to enable replication, pick peers, and recover from failures. Replication is enabled for every paid Gravwell license, but Community Edition licenses cannot enable replication.

WARNING: The `Indexer-UUID` variable in the gravwell.conf file is used as the indexer identity in the replication group. When restoring an indexer, it is critically important that you restore the correct UUID prior to starting the indexer.

5.4.1 Offline Replication Configuration

Replication is entirely controlled by indexers. While webservers are aware of indexer configurations and can request hot failover during search, they do not have control over or insight into replication state or peer selection. To enable replication, we add a replication configuration block to the indexer's `gravwell.conf` file. Below is an example replication configuration with a single offline replication peer:

```
[Replication]
    Peer=172.17.0.3
    Storage-Location=/opt/gravwell/replication_storage
    Disable-TLS=true
    Connect-Wait-Timeout=60
    Disable-Server=true
```

Notice that we disabled the server, but still specified a Storage-Location; replication needs to store the state of replicated data. While our server may not be replicating data from other nodes, it still needs to track how much of its own data has been replicated.

There are a few important security considerations associated with the above config:

1. `Replication-Secret-Override` is not specified.
   (a) The indexer will use the `Control-Secret` for authentication.
2. TLS is disabled, which means that indexer is using a cleartext connection.
   (a) Data is replicated in the clear and restored in the clear
3. `Connect-Wait-Timeout=60` means Indexers will wait up to 60 seconds for a replication peer.
   (a) If an indexer cannot connect to a replication peer in that time, it will start.
   (b) This is important when recovering from a failure.
   (c) When restoring a failed node, it is recommended that this setting be 0, meaning the indexer will wait for a replication peer indefinitely before starting.

5.4.2 Hands-on Lab: Replication

We will be instantiating an offline replication server and configuring our test container to replicate data to it. The offline replication server acts as a remote data store of replicated data and will allow our indexer to recover from failure and re-import its data. After replicating the data from our test instance we will destroy it and start over. When the indexer comes back online, it will identify the failure and restore its data and tag set from the replication server.

Starting the Offline Replication Server

Change your current working directory to `gravwell_training/Indexers/Lab-Replication` and check its contents. You should see a gravwell.conf file in the `config/` subdirectory. Make sure you've got the `offlinereplication` image loaded as described in Section 2.5, then start it with a name and hostname of `offlineserver`:

```
docker run -d --net gravnet --name offlineserver \
-h offlineserver gravwell:offlinereplication
docker ps
```

Next, stop your test container and edit the gravwell.conf to include a replication configuration block so that it can replicate its data to the offline replication server. We can use the hostname `offlineserver` for the replication peer and leave the authentication secrets blank for reasons that will be covered in section 5.7. Once you are happy with your replication configuration block, copy it into the test container and restart it. Then use ping and netstat to verify that the indexer has connected to the offline replication server:

```
docker cp config/gravwell.conf test:/opt/gravwell/etc/
docker restart test
docker exec -it test ping -c 1 offlineserver
docker exec -it test netstat -pa | grep ESTABLISHED
```

We should see an established connection to the host `offlineserver` on port 9406 (the replication port). Get a shell on the offline replication server using `docker exec -it offlineserver /bin/sh`. Print the replication configuration file to find the replication storage directory, then navigate to that directory. We should see a `replication.db` file and a directory with the same name as our indexer's UUID. This means the indexer has connected and is pushing shards to the offline replication server. Wait a few minutes and watch as the indexer pushes each successive data shard to the replication server.

Once the indexer has pushed all its shards (you should see about 20), exit the offline replication server shell with the `exit` command.

Next, let's simulate a disk failure by destroying all the shards in our indexer and restarting it. Let's start smashing some storage locations within our indexer, then restart it:

```
docker exec test rm -rf /opt/gravwell/storage/
docker exec test rm -rf /opt/gravwell/cold_storage/
docker restart test
```

These docker commands basically reached in and annihilated all the data in our indexer and then restarted the entire container. If not for the replication system we would have lost everything. However with replication, if we wait just a moment we can see the indexer repopulate its shards, ready for searching. You may have noticed that the replication system restored the shards to the cold pool. The replication system will typically target cold pool for restoration, only pushing current hot shards to the hot pool.

Lab Questions

1. Did the shard size change on the indexer after it was restored?
   (a) Why?
2. Why would we want to restore shards to the cold well on replication recovery?

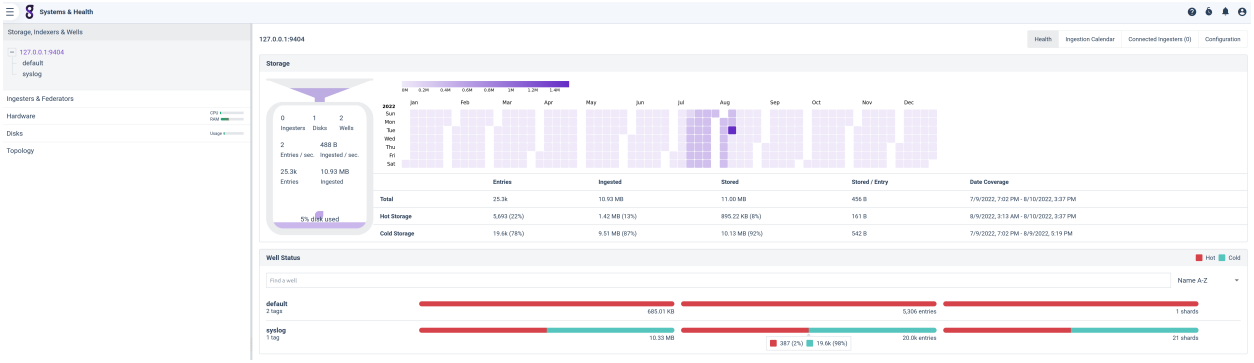


Figure 5.6: Post-restoration Well Stats

To clean up after the experiment, simply run:

```
docker stop $(docker ps -a -q)
docker rm $(docker ps -a -q)
```

5.5 Query Acceleration and Indexing

Gravwell indexers provide a couple different data management methodologies, ranging from raw storage with only temporal indexing to full feature extraction with direct data indexing. This section will explore indexer acceleration configuration and examine some of the pros and cons of this methodology.

A Gravwell well without any acceleration configuration will employ only temporal indexing, which means that every entry is grouped according to a timestamp that is indexed using a temporal index. The temporal index allows for specifying subsections of time without combing through data that isn't in the time region specified by the query. Wells can also be configured to enable a secondary index which takes into account data contents. The secondary indexes use feature extraction modules which behave similarly to query modules.

The feature extraction modules are responsible for extracting fields from raw entries and passing them to an acceleration engine. The acceleration engine then generates a data structure which can help queries efficiently filter and lookup specific data features. Gravwell currently supports two acceleration engines: bloom and index.

The two acceleration engines are designed to provide a tradeoff between disk usage and query speed. At low data volumes where an indexer can easily cache an entire bloom index, the bloom engine can provide an efficient method for query acceleration. When data volumes are large enough that it is not practical to cache significant portions of the index in memory, the index engine allows for direct disk-backed query acceleration.

Note that the acceleration engines will only be engaged when using the equality (==) filter, because the engines index exact values. Filters such as less-than, great-than, or subset will not be accelerated, because they require more complex comparisons; they can still be used and will work correctly, but the acceleration engines will not be engaged for those filters and the query will not execute any faster than on non-accelerated data.

5.5.1 Accelerator Well Configuration

Accelerators are configured on a per well basis, which enables tailoring field extraction and acceleration engines to specific data. Practitioners can choose the appropriate storage and query performance tradeoffs for each data type based on retention requirements, storage availability, and data query frequency. Very large deployments with many hundreds of terabytes of data can achieve significant cost reductions by choosing the appropriate acceleration and storage mechanic for each data source.

Let's examine a very basic acceleration configuration which expects JSON data and extracts a few fields. The two required configuration parameters are `Accelerator-Name` and `Accelerator-Args`. The `Accelerator-Name` parameter defines which field extraction module is responsible for processing incoming data, and the `Accelerator-Args` parameter defines what the module should extract.

An example configuration which extracts a username, email, country, group, and ip from a JSON data stream might look like:

```
[Storage-Well "json"]
    Tags=json
    Location=/opt/gravwell/storage/json/
    Accelerator-Name="json"
    Accelerator-Args="user email country group ip"
```

The acceleration configuration does not specify an `Accelerator-Engine-Override` which means the default engine `bloom` is used. If we were to add `Accelerator-Engine-Override=index` to the configuration the acceleration system would use direct indexing rather than a bloom filter for the engine. The direct indexing system would consume more storage space, but may also be significantly faster for large data sets.

Entry source fields can also be added to the acceleration configuration to allow the `src` module to also invoke acceleration. Accelerating on source may be useful when there are many ingest endpoints and you want to only look at data that was generated by a specific source. Enabling entry source extraction is performed by adding `Accelerate-On-Source=true` to the well configuration. Source acceleration is entirely independent of the extraction module. Other acceleration tuning parameters such as the bloom collision rate are also available; make sure to visit the official Gravwell accelerator documentation page for a full list of parameters and tuning options.

Let's examine some entry data where the above acceleration configuration would be applicable:

```
{
"time":"2019-03-18T17:00:22.648353602-06:00",
"class":13897,
"user":"madisonmartinez880",
"name":"Elijah Taylor",
"email":"madisonmartinez880@test.com",
"state":"DC",
"country":"Monaco",
"group":"toucan",
"useragent":"Mozilla/5.0 AppleWebKit/537.36 (KHTML, like Gecko)
Chrome/60.0.3112.107 Mobile Safari/537.36",
"ip":"6.89.189.142"
}
```

The example data is something that might be generated by a service API that responds to user requests and logs various data about the user. Our accelerator configuration is setup to extract some specific fields from the entry to enable query acceleration. It is important to note that the field extraction system is much more tolerant in the acceleration system than it is in query. For example, if the field `user` was missing from the entry, the `json` query module would drop the entry; in the acceleration system entries are never dropped and field extraction modules will continue attempting to extract any specified fields.

Invoking the acceleration system is entirely transparent. Users need only specify inline filtering arguments at query time and the indexers will identify whether a field is accelerated on and utilize it. Let's look at a query that would invoke acceleration using our example configuration:

```
tag=json json user country=="Monaco" useragent~"Apple" | table
```

The query is performing a few inline filters to look for specific field values and then outputting the resulting enumerated values into a table. Three fields are extracted, with a filter applied to the `country` and `useragent` fields. The `country` field has an equality filter, specifying that the extracted `country` must be the exact

Engine	Storage Used	Storage %	Ingest Rate	Ingest %
none	N/A	N/A	145K/s	100%
bloom	110MB	6.4%	100K/s	68%
index	770MB	45%	61K/s	42%

Table 5.1: Storage Cost by Acceleration Engine

Engine	Query Time	Speedup
none	6.32s	N/A
bloom	700ms	9X
index	220ms	28X

Table 5.2: Speedup by Acceleration Engine

value "Monaco" and the `useragent` filter simply says that the value "Apple" must occur somewhere within the `useragent` field. While both the `name` and `country` fields are specified in the acceleration config, only the country field has an appropriate filter operation, so the acceleration system will only use the country field for acceleration. The `useragent` field does not use a supported filter operator; ~ is a subset operation and can't be used for acceleration, because the acceleration system can only compare against complete matches, not substrings.

Accelerator configurations can be changed at any time, however indexers will only apply the new accelerator configuration on new shards, old shards maintain their old configuration. This means that changing accelerator configurations does not cause Indexers to reindex old data. The query system will re-evaluate the applicability of a query for acceleration on each shard. For example, if we added the field `group` to our acceleration configuration after a few days of operation, only the new shards would have the field in their indexes. Any query that specified an equality filter on the `group` field will transparently use the group field on shards which have it and not on shards that do not. Users may notice speedups on specific time spans of data but otherwise don't need to care what is and is not accelerated.

Gravwell supports an ever growing list of field extraction modules for acceleration, including: csv, fields, syslog, json, cef, regex, winlog, and binary slicing. Check the official Gravwell documentation page at https://docs.gravwell.io/#!configuration/accelerators.md for an up-to-date list of supported extraction modules.

5.5.2 Accelerator Overhead and Query Impact

Enabling acceleration on Gravwell indexers can have a profound impact on query performance. A well-tuned index can enable a query to find a few specific entries in terabytes of data in just a few milliseconds. However, indexing is not free; it incurs disk and ingest rate costs. This section will explore the resource costs for each acceleration engine type to help you better understand when to use them. Let's start by comparing the impact acceleration has on a sample dataset consisting of 10 million JSON entries. The JSON dataset is approximately 5GB of data and consumes about 1.7GB of space when stored on an indexer using the default compression system. We can see the impact of just temporal indexing vs bloom filtering vs direct indexing for each engine in Table 5.1.

However, if we look at the impact on query performance using each acceleration engine we can see how the storage and ingest impacts are warranted. Table 5.2 shows the query speedup of the different acceleration engines; note that although the index engine requires a lot of storage space to hold the index, it offers a 28x speedup over un-accelerated querying.

The example dataset and system allowed the entire bloom index to be held in memory during query, but the direct index system was still significantly faster. Systems where the bloom index cannot be held in memory will see even more dramatic speedups. One Gravwell customer saw a query that examined 8TB of data complete in less than 1 second once indexing was properly configured, where un-accelerated queries had previously taken over 30 minutes.

5.5.3 Accelerators and Query Modules

Many query extraction and some filtering modules can make use of query accelerators (like full text indexing, JSON indexing, etc) when filtering is used with a given module. For example, the netflow module (`netflow Src Dst Port==22`) can use a properly-configured accelerator to dramatically reduce search time because not all records need to be evaluated. Some filtering modules (such as "words") can also invoke query acceleration by passing hints to the underlying accelerator.

Not all query modules are compatible with all query acceleration configurations. For example, if the "netflow" tag is configured to be accelerated using the "netflow" accelerator, the "words" module will not be able to invoke query acceleration on this tag; this is because the "netflow" accelerator is expecting to operate on binary data and apply a specific structure to data during indexing, while the "words" module needs fulltext acceleration. Gravwell will intelligently examine query parameters and invoke the acceleration system whenever possible, but there are some caveats to be aware of.

Query acceleration that uses specific structure (such as netflow, ipfix, packet, fields, regex, etc) requires that the query parameters match exactly.

For example, if you customize your accelerator for a given tag so that it uses regular expressions to apply structure, then you must use the exact regular expression in your query in order to benefit from the accelerator. If the regular expression does not match exactly, the system will not engage query acceleration, because it cannot guarantee that the query regular expression is a direct subset of the accelerator regular expression.

Query acceleration may accelerate on subsets of your query.

For example, let's assume the tag "test" is using fulltext acceleration and we are executing the query `tag=test grep "this that the other"`. The query is using grep as a brute-force sub-string match which means that the boundaries of the match string are not necessarily word boundaries. However, the query system is smart enough to know that some internal parts of the matched string are words, so it will use them to accelerate the query. The query will accelerate the search as if you had run `tag=test words that the | grep "this that the other"`.

Query acceleration is not order of operation dependent.

Modules which can accelerate queries will hint about their ability to accelerate no matter where they are in the query string. Consider the following query: `tag=test grep foo* | regex "my name is (?P<name>\S+)" | words foobar`. The `words` module is capable of invoking query acceleration on a fulltext accelerator and will hint to the acceleration system that it wants the word "foobar". You may notice that very few entries actually enter the pipeline even though the `words` module came much later in the query.

Query acceleration is on a shard-by-shard basis.

Gravwell does not require tag accelerators to be consistent across all time. You can set up acceleration, ingest some data, and then change the acceleration configuration without re-indexing data. When you issue a query, the acceleration hits are handed to each shard of data across time and the compatibility of acceleration is checked on each shard. Gravwell will automatically invoke acceleration wherever possible, which means that as your query moves over historical data it may be engaging acceleration in different ways transparently. You may notice that a query is fast on some sections of data and slower on others–that is just the system engaging acceleration where it can.

Query acceleration operates on positive matches.

Gravwell is not a bitfield index; this means that it will only accelerate on direct matches. For example `tag=syslog syslog Hostname=="foobar"` can invoke the accelerator, but `tag=syslog ProcID < 20` will not. The accelerators also do not accelerate on the negative, meaning that `tag=syslog syslog Hostname != "foobar"` will not invoke the accelerator engine.

5.5.4 Hands-on Lab: Acceleration

For this lab, we are going to configure a well that will store JSON data and enable an accelerator that will extract fields from the JSON data at ingest time. We will then generate hundreds of megabytes of data using the JSON generator. Once the data is ingested we will query it and examine how the accelerator reduces the number of entries that enter the pipeline, thus speeding up the query.

First, we'll get into the lab subdirectory:

```
cd gravwell_training/Indexers/Lab-Acceleration/
```

Create a new container using the gravwell:base image:

```
docker create --net gravnet --name test gravwell:base
```

We have provided a copy of the base gravwell.conf in the config/ subdirectory, so modify it to create a new well named "json" with the following specifications:

1. Storage location of `/opt/gravwell/storage/json`
2. Tag `json` assigned
3. `Accelerator-Name="json"`
4. Extract the following fields:
   (a) `class account.user account.email account.phone account.state account.country group ip`

Copy the edited gravwell.conf file into the container, start it, and inspect it to find its IP:

```
docker cp config/gravwell.conf test:/opt/gravwell/etc/gravwell.conf
docker start test
```

Now we'll use the generators image to generate some JSON data; if you don't have the `gravwell:generators` image, see Section 2.5 for instructions on how to load it.

```
docker run --net gravnet --rm -it gravwell:generators \
/jsonGenerator -clear-conns test:4023 -entry-count 500000
```

Open your Gravwell GUI and check that there is a new well named "json" with 500k entries in it. Then execute a query over the last week which uses the `json` module and inline filtering to display only entries where the `account.country` field is "Greece":

```
tag=json json account.country==Greece
```

You should see a few thousand entries, as shown in Figure 5.7

Now click the stats tab on your results page to view the search stats, as shown in Figure 5.8. Pay special attention to the "Entries processed" value. That value specifies how many entries actually entered the pipeline. Your dataset contained 500k entries, but using the inline filtering and query accelerators enabled you to only have to process a few thousand entries. That's hundreds of thousands of entries that we didn't have to pull off the disk or look at in any way.

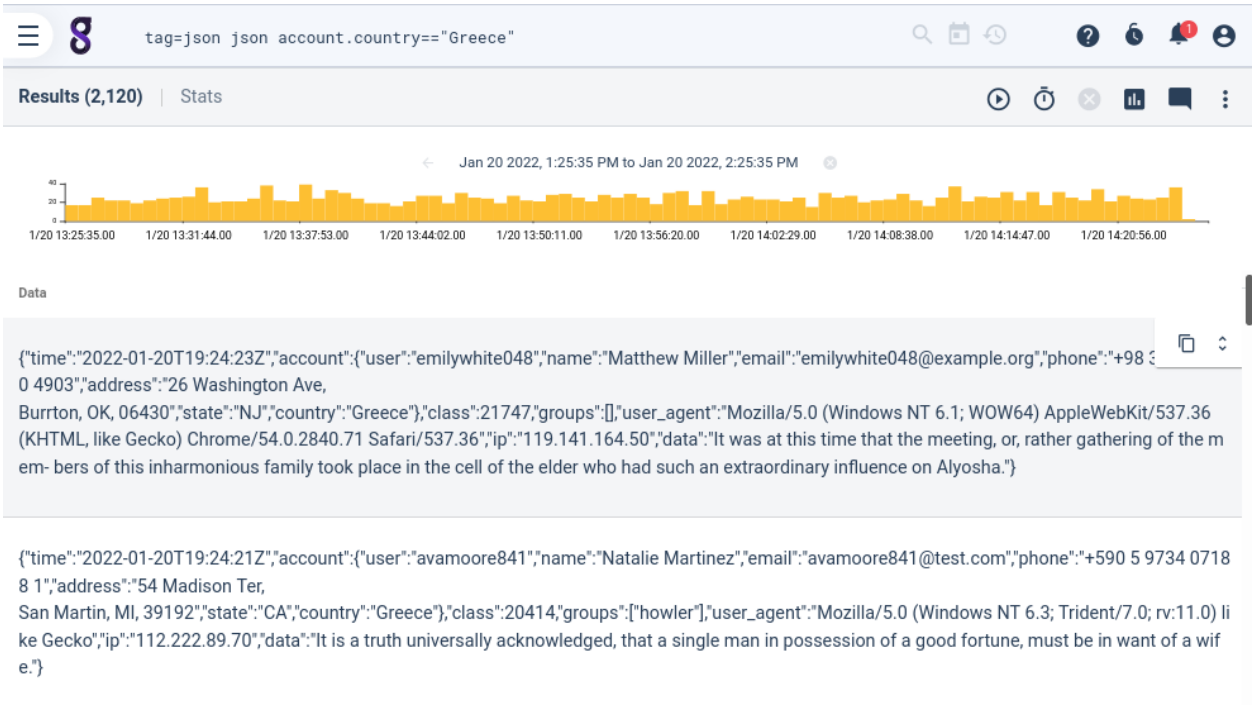


Figure 5.7: Filtered Search Results

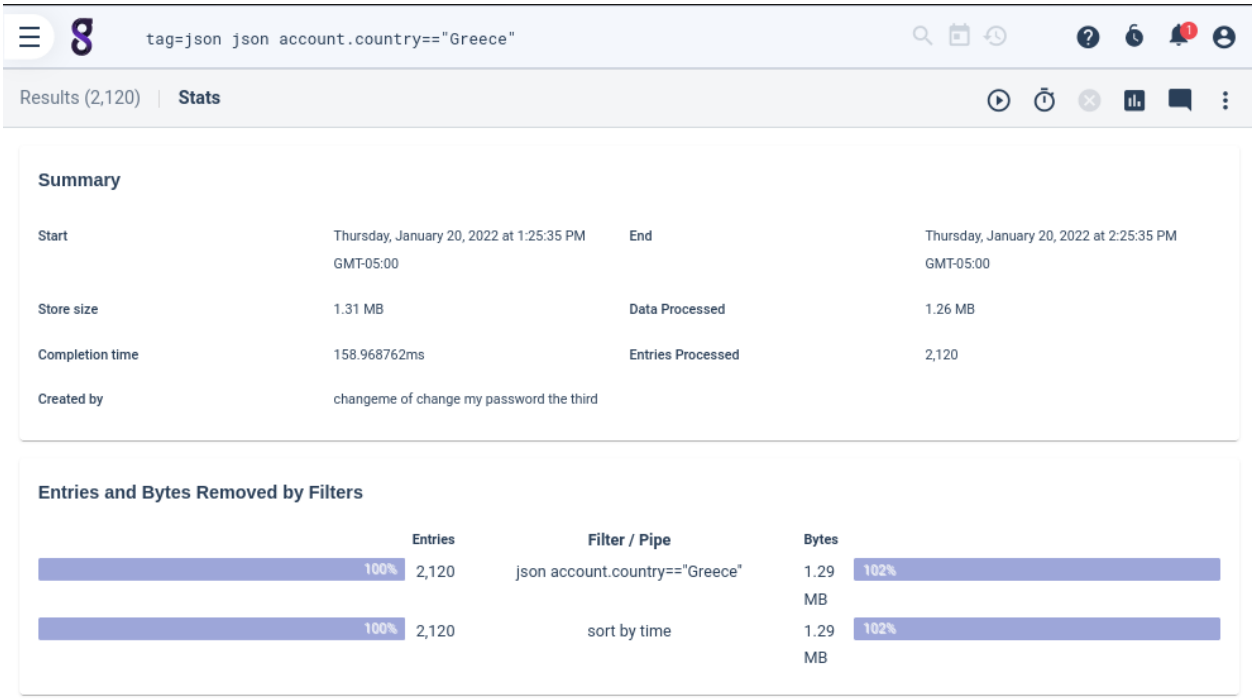


Figure 5.8: Search Results Stats

Next, we'll add two additional wells to the indexer. The second well will enable query acceleration that is almost identical to the well we have already configured, but instead of using the default bloom engine, we are going to use the index engine. The third well will not enable query acceleration at all.

Open your gravwell.conf file and add two additional wells with the following parameters:

1. Name: `json2`
   (a) Storage location `/opt/gravwell/storage/json2`
   (b) Tag `json2` assigned
   (c) `Accelerator-Name="json"`
   (d) Extract the following fields: `class account.user account.email account.phone account.state account.country group ip`
   (e) Set engine via `Accelerator-Engine-Override="index"` parameter
2. Name: `json3`
   (a) Storage location `/opt/gravwell/storage/json3`
   (b) Tag `json3` assigned

Stop your gravwell container, copy the updated `gravwell.conf` file into it, and then restart it:

```
docker stop test
docker cp config/gravwell.conf test:/opt/gravwell/etc/gravwell.conf
docker start test
```

Re-ingest more JSON data using the tags "json2" and "json3":

```
docker run --net gravnet --rm -it gravwell:generators \
/jsonGenerator -clear-conns test -entry-count 500000 -tag-name json2
```

```
docker run --net gravnet --rm -it gravwell:generators \
/jsonGenerator -clear-conns test -entry-count 500000 -tag-name json3
```

Go re-run the same query as before over a week, first using the tag "json2" then again using the tag "json3":

```
tag=json2 json account.country==Greece
tag=json3 json account.country==Greece
```

You should get a few thousands results for each query, but check the stats page and examine the "Entries Processed" results. For the tag `json3`, which used a well that had no query acceleration, you should notice that the query had to actually process 500k entries. That means the indexer had to get the 500k entries off the disk and put them in pipeline. The `json` module was then responsible for performing all the filtering, meaning it had to process every single entry. For the `json2` tag, you should notice that the number of entries processed was the exact same as the number of results. That is because the `index` engine doesn't have a high statistical collision rate the way the `bloom` engine does. By reading even fewer entries off the disk than with the bloom engine, this query is even faster.

Lab Questions

1. How much more storage did the bloom engine well consume than the raw well?
2. How much more storage did the index engine well consume than the bloom well?
3. What was the difference in ingest speed for each well?
4. How much faster was the index engine well than the bloom well?
5. And how much faster was the bloom well than the non-accelerated (raw) well?
6. If we re-run each query multiple times, how much faster was each?

(a) Why?

(b) When would this NOT happen?

To clean up after the experiment, simply run:

```
docker stop $(docker ps -a -q)
docker rm $(docker ps -a -q)
```

5.6 Indexer Optimization

Gravwell prides itself on not requiring specific machine specs and being able to scale across a broad range of hardware capabilities. While the software makes a best effort at scaling without human intervention, there are several tuning parameters that can improve storage and query efficiency on very large machines, and protect very small machines from memory exhaustion. The tuning parameters are focused on how the indexers organize discrete units of data and how many of those units can be "in flight" at any given time. Varying each parameter allows machines with lots of CPU cores and significant memory to scale out and feed those cores as well as make more efficient use of storage by effectively grouping data. Most default values are geared towards a sane default that will operate well on a smaller machine.

As data is ingested into a Gravwell indexer it is grouped in storage units called blocks. The more efficiently that like data can be colocated into blocks, the more efficient we can store and query data. Gravwell indexers allow for fine tuning maximum block sizes and the facilities used to generate those blocks.

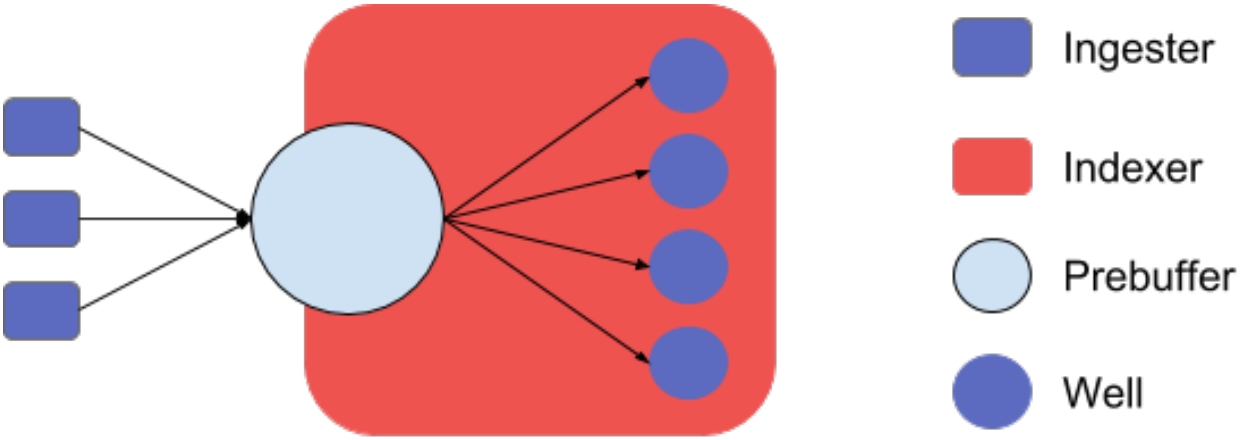


Figure 5.9: Indexer Prebuffer

Each indexer employs a prebuffer (Figure 5.9) used during ingest to hold a set of incoming entries and organize them for better storage efficiency. The prebuffer allows an indexer to tolerate ingesters that are sending out-of-order data, or multiple ingesters with clock skews, while still generating relatively efficient storage blocks. Large machines with significant memory can allocate a larger prebuff to enable faster ingest and better storage efficiency, while smaller machines with limited memory can reduce the prebuffer so that more memory is available for query.

Tuning block sizes is typically not required, but if you are operating a very large cluster with very high data ingest rates, tuning the block sizes can help query performance, compression efficiency, and ingest throughput. By default indexers will store up to 4MB of data in a single block. This means that if more than 4MB of data with the same timestamp is ingested, it will be broken into multiple blocks. The maximum block size is controlled using the `Max-Block-Size` parameter in the global section of the `gravwell.conf` file. The size of a block of can impact memory usage at query time, storage efficiency, and compression efficiency. However, if you specify very large blocks and large pipeline buffers, your indexer will consume significantly more memory at query time and may not tolerate large numbers of concurrent queries.

5.7 Docker Configuration

Throughout the training so far we have made heavy use of Docker as a simple container platform that makes it easy to configure and run Gravwell. As you were working with Gravwell, there may have been a nagging question about the gravwell.conf files we were using. There were no secret tokens in any of them. The docs clearly state that you MUST set the `Ingest-Secret` and `Control-Secret` parameters, but none of the gravwell.conf files we have used so far have done so. Why? How have things worked at all?

Docker is designed to rapidly deploy services with as little fanfare as possible. Kubernetes and RedHat OpenShift build upon that concept and take it to the extreme, making it possible to deploy extremely large clusters of services with just a few commands. Configuring those services with thousands of configuration files would be ridiculously difficult, so the platforms provide a system called "secrets" which allow you to inject environment variables and secure tokens into a container at runtime. Those injected variables are how we have been configuring the secrets: they were registered with the image and injected every time you fired up a container. Check it by inspecting a running gravwell:base container or image:

```
docker inspect gravwell:base
```

The pertinent section is `Config.Env`, which should contain the following environment variables:

```
GRAVWELL_INGEST_AUTH=IngestSecrets
GRAVWELL_INGEST_SECRET=IngestSecrets
GRAVWELL_CONTROL_AUTH=ControlSecrets
GRAVWELL_SEARCHAGENT_AUTH=SearchAgentSecrets
GRAVWELL_PIPE_TARGETS=/opt/gravwell/comms/pipe
```

Gravwell components will attempt to read configuration variables from the `gravwell.conf` file, but if required variables are not present in the config file it will look for them in the environment, and optionally through Docker secrets[4]. Gravwell will only look at environment variables and/or secrets if the corresponding gravwell.conf parameter is empty; the configuration file always supersedes any environment or secret tokens. For more information see our documentation section on Docker configuration[5]

The environment variables associated with the image can be overridden at runtime using the `-e` Docker flag. Remember that environment variables are NOT a secure way to configure Gravwell, so if you are going to use Docker, Kubernetes, or Openshift in production, use secrets for the auth tokens.

[4] https://docs.docker.com/engine/swarm/secrets/
[5] https://docs.gravwell.io/#!configuration/docker.md

5.7.1 Hands-on Lab: Docker Configuration

For this lab we are going to use Docker environment variables to stand up a distributed 4-node Gravwell cluster in just a few commands without fiddling with configuration files. We will inject all the needed configuration parameters using Docker. If you are using a community license you can complete the lab, but will only be able to configure a single indexer.

Look at `gravwell_training/Indexers/Lab-Docker/config/gravwell.conf` and note that we have removed the `Remote-Indexers` configuration parameter so that we can inject a list of indexers at runtime. First, fire up a container that is only running the indexer, show the process list, and then clean up the container:

```
docker create --net gravnet --name idx \
    -e DISABLE_WEBSERVER=TRUE -e DISABLE_SEARCHAGENT=TRUE \
    gravwell:base
docker cp gravwell.conf idx:/opt/gravwell/etc/
docker start idx
docker exec idx ps
docker stop idx
docker rm idx
```

You should see that only the `manager` and `gravwell_indexer` processes are running. The manager application is an open-source[6] Gravwell process management application that somewhat takes on the role of systemd. It respects some configuration via environment variables of its own.

```
user@training:~$ docker exec -it idx ps
PID   USER     TIME  COMMAND
    1 root      0:00 /opt/gravwell/bin/manager
   11 root      0:00 /opt/gravwell/bin/gravwell_indexer -stderr indexer
   22 root      0:00 ps
user@training:~$
```

Now that we know how to start an indexer in a container without the other services, let's fire up a few of them with a small loop. Make sure you're in the `Indexers/Lab-Docker/config` directory, then run:

```
for i in 1 2 3 4; do
    docker create --net gravnet --name idx$i \
    -e DISABLE_WEBSERVER=TRUE \
    -e DISABLE_SEARCHAGENT=TRUE \
    gravwell:base
docker cp gravwell.conf idx$i:/opt/gravwell/etc/
docker start idx$i
done
```

We should now have 4 indexers up and running, so let's start up the webserver and configure it to connect to our indexers:

```
docker create --net gravnet --name webserver \
    -e DISABLE_INDEXER=TRUE \
    -e GRAVWELL_REMOTE_INDEXERS=idx1,idx2,idx3,idx4 \
gravwell:base
docker cp gravwell.conf webserver:/opt/gravwell/etc/
docker start webserver
```

Get the webserver's IP address:

```
docker inspect --format \
'{{.NetworkSettings.Networks.gravnet.IPAddress}}' webserver
```

[6] https://github.com/gravwell/gravwell/tree/master/manager

Let's open up our Gravwell GUI and check out our indexer and hardware tabs. You should see several indexers connected with a lot more activity in the hardware monitoring page, as shown in Figures 5.10 and 5.11.

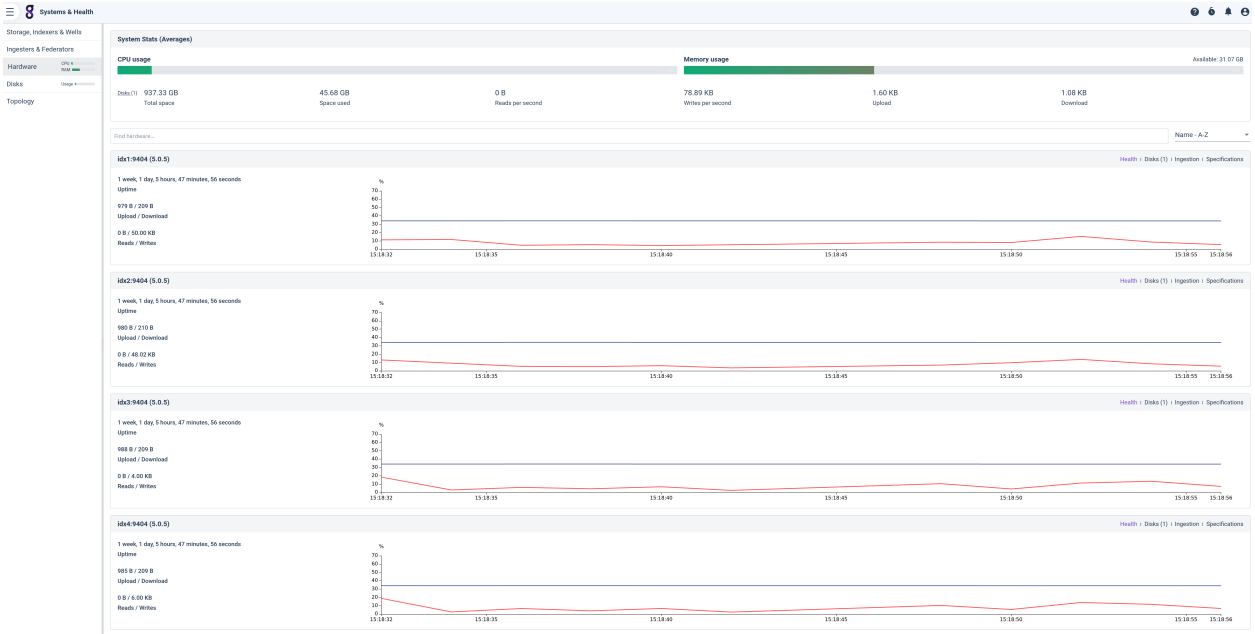


Figure 5.10: Hardware Page with Four Indexers

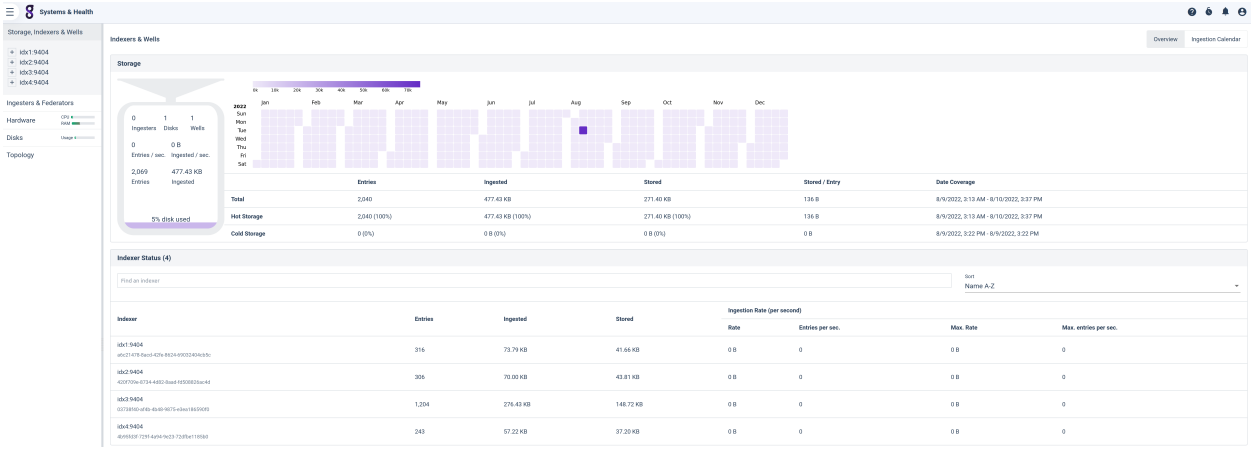


Figure 5.11: Wells Page with Four Indexers

To clean up after the experiment, simply run:

```
docker stop $(docker ps -a -q)
docker rm $(docker ps -a -q)
```

Chapter 6

Webserver Configuration

Gravwell's webserver component provides users with access to Gravwell's search capabilities. The simplest system consists of one indexer and one webserver, both on the same machine. More complex variations are possible, such as one webserver and multiple indexers or even multiple webservers with one or more indexers. This chapter will discuss configuration options for the Gravwell webserver.

6.1 Basic Configuration

The webserver is configured via `gravwell.conf`. The following basic options can be set in gravwell.conf, but defaults are usually functional:

- `Web-Port` (default 443): the port on which the webserver should listen.
- `Insecure-Disable-HTTPS` (default false): enabling this makes the webserver operate HTTP-only.
- `Key-File` (default /opt/gravwell/etc/key.pem) and Certificate-File (default /opt/gravwell/etc/cert.pem): specify an X509 pair to use for HTTPS
- `HTTP-Proxy` (default none): the address of an HTTP proxy that should be used for requests to Internet resources

The following settings control some parameters around login sessions and brute-force protection. The defaults are typically acceptable:

- `Session-Timeout-Minutes` (default 60): how many minutes a login session should last without activity
- `Login-Fail-Lock-Count` (default 5): how many failed login attempts before a user account is locked
- `Login-Fail-Lock-Duration` (default 5): how many minutes a user account should remain locked after a brute-force attempt

6.1.1 Configuring Indexers

A webserver must connect to one or more Gravwell indexers to issue queries, since the actual data resides on the indexers. The default community edition configuration only talks to one indexer:

```
Remote-Indexers=net:127.0.0.1:9404
```

Adding more indexers is as simple as specifying additional Remote-Indexers entries in gravwell.conf:

```
Remote-Indexers=net:indexer0:9404
Remote-Indexers=net:indexer1:9404
```

Gravwell will automatically use all indexers it knows about when searching. No additional configuration is required on the indexers.

6.1.2 Hands-on Lab: Adding Indexers to a Webserver

In this lab, we will take an existing webserver + indexer pair and configure the webserver to also use an additional indexer.

First, verify that you have the gravwell:base and gravwell:indexer images, if not load them from your images directory as detailed in Section 2.5. Next we will launch two containers. The first is a webserver and indexer on the same system, the second is just an indexer:

```
docker run --rm -p 8080:80 -d --net gravnet --name webserver gravwell:base
docker run --rm -d --net gravnet --name indexer0 gravwell:indexer
```

Next, we point a web browser at http://localhost:8080 and log in.

Then, we add the new indexer to the webserver's configuration. Copy gravwell.conf from the webserver to the temp directory:

```
docker cp webserver:/opt/gravwell/etc/gravwell.conf /tmp
```

Edit `/tmp/gravwell.conf` and add the following line below the existing 'Remote-Indexers' line:

```
Remote-Indexers=net:indexer0:9404
```

Save the file, then run the following to re-update the file to the webserver:

```
docker cp /tmp/gravwell.conf webserver:/opt/gravwell/etc/gravwell.conf
```

Finally, we restart the webserver container:

```
docker restart webserver
```

Now, browsing to the "Wells and Indexers" page in the GUI should show two indexers, as in Figure 6.1

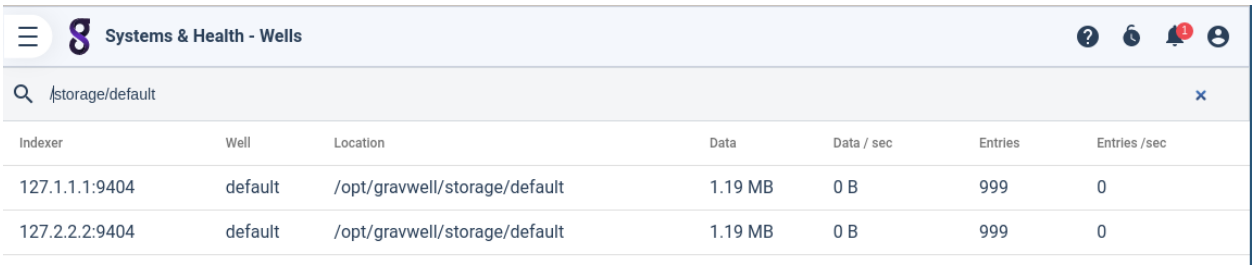


Figure 6.1: Two indexers

Lab Questions

1. Why was it necessary to restart the webserver process after updating the configuration?
2. How would you configure the webserver to *only* communicate with the indexer on the indexer0 container, not the local instance?

To clean up after the experiment, simply run:

```
docker kill $(docker ps -a -q)
```

6.2 Configuring Multiple Webservers

Gravwell can use multiple webservers to load-balance user requests. These webservers must coordinate with each other to manage users, searches, etc. This coordination is handled through the **datastore**, a centralized authority for webserver data.

Setting up a multiple webserver environment comprises configuring the datastore, pointing the webservers at the datastore, and optionally installing a load-balancer to balance traffic between the webservers.

The datastore is a separate Gravwell component which must be installed via self-extracting shell archive available on the Gravwell downloads page. The datastore can be co-resident with a Gravwell webserver, or it can run on its own devoted machine. It reads its configuration from gravwell.conf; for most situations, defaults will be fine, but the following options are available:

- `Datastore-Listen-Address` (default "" [all]): specifies which IP address the datastore should listen on for connections
- `Datastore-Port` (default 9405): specifies the port on which the datastore should listen
- `Control-Auth`: the shared secret by which webservers authenticate to the datastore. This must be the same on the datastore and webservers!

Webservers must in turn be configured to speak with the datastore using the following options:

- `Datastore`: the address and optional port to connect to the datastore, e.g. "`datastore.gravwell.io:9405`".
- `External-Addr`: The address which external systems should use to access *this webserver*. It can be an IP address or a DNS name. Setting this value allows users on webserver A to view searches on webserver B.
- `Datastore-Update-Interval` (default 10): how often (in seconds) the webserver should check in with the datastore. 10 is a good default.
- `Datastore-Insecure-Skip-TLS-Verify`: if set to true, the webserver will not verify the datastore's TLS certificate. You should never use this in production, but it can be useful for testing.

After configuring the webserver's gravwell.conf, restart the webserver process. It should connect to the datastore and begin synchronizing. Repeat this for all webservers

6.2.1 Hands-on Lab: Configuring multiple webservers

This lab will configure an indexer, two webservers, and a datastore. Note that for simplicity, we will be using self-signed TLS certificates for communications between the webservers and the datastore and will therefore be disabling TLS verification in our configs. For actual deployments we strongly recommend configuring properly-signed TLS certificates as explained in Section 14.1.

First, verify that you have the gravwell:webserver, gravwell:indexer, and gravwell:datastore images available. If not, load them from your images directory as detailed in Section 2.5.

Next, we start four containers: one indexer, two webservers, and the datastore. Note that webserver0 is forwarded to localhost:8080, and webserver1 is forwarded to localhost:8081:

```
docker run --net gravnet --rm -d --name indexer0 gravwell:indexer
docker run --net gravnet --rm -p 8080:80 -d --name webserver0 gravwell:webserver
docker run --net gravnet --rm -p 8081:80 -d --name webserver1 gravwell:webserver
docker run --net gravnet --rm -d --name datastore gravwell:datastore
```

Next, we configure each webserver to talk to the datastore. Do the following for both webserver0 and webserver1, taking care to set the `External-Addr` field properly for each:

Copy the config file to the local system (e.g. `docker cp webserver0:/opt/gravwell/etc/gravwell.conf /tmp/gravwell.conf`) and add the following in the `[Global]` section:

```
Remote-Indexers=net:indexer0:9404
Datastore=datastore
External-Addr=webserver0
Datastore-Insecure-Skip-TLS-Verify=true
```

Copy the config file back (`docker cp /tmp/gravwell.conf webserver0:/opt/gravwell/etc/gravwell.conf`) and restart the webserver:

```
docker restart webserver0
```

Now we repeat the process for webserver1; **note that the External-Addr field is changed below!**

Copy the config file to the local system (e.g. `docker cp webserver1:/opt/gravwell/etc/gravwell.conf /tmp/gravwell.conf`) and add the following in the `[Global]` section:

```
Remote-Indexers=net:indexer0:9404
Datastore=datastore
External-Addr=webserver1
Datastore-Insecure-Skip-TLS-Verify=true
```

Copy the config file back (`docker cp /tmp/gravwell.conf webserver1:/opt/gravwell/etc/gravwell.conf`) and restart the webserver:

```
docker restart webserver1
```

Now connect to http://localhost:8080 and http://localhost:8081 in two separate tabs. Log in ("admin"/"changeme") to both, leaving each open in its own tab.

Open the Groups management screen in the menu bar on both webservers as shown in Figure 6.2.

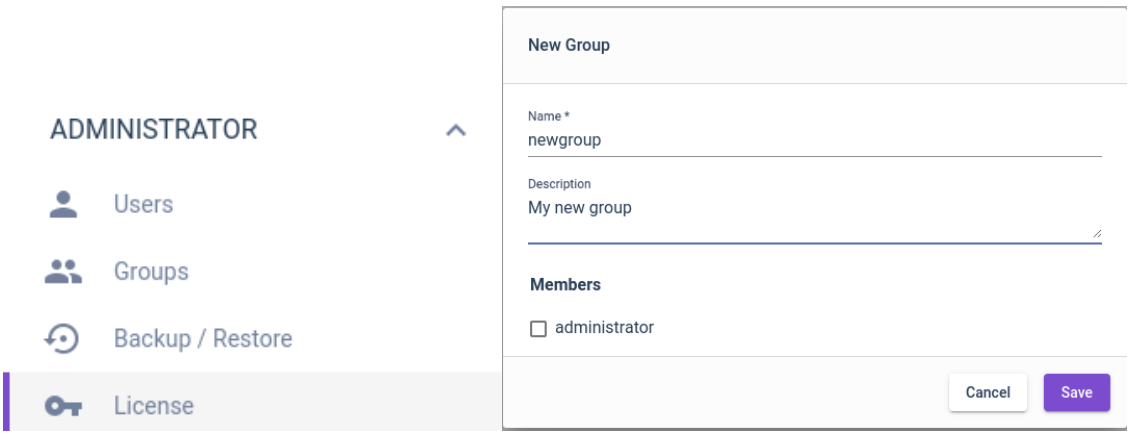


Figure 6.2: Adding a new group

On one webserver, create a new group; before long, that new group should be visible on the other webserver's Groups screen too. It may be necessary to refresh the page.

Lab Questions

1. Why does it take some seconds for the new group to appear on the second webserver? How might this be sped up?

To clean up after the experiment, simply run:

```
docker kill $(docker ps -a -q)
```

6.3 Setting Up a Load-Balancer

Multiple webservers work well when placed behind a load-balancing proxy, such as nginx, Traefik, or Gravwell's own load-balancer. When using a load-balancer, users simply access the load-balancer's address (e.g. gravwell.example.org) and are transparently proxied through to one of the webservers (e.g. gravwell-webserver-01.example.org).

It is essential that the load balancer be configured with "sticky" sessions. This ensures that a given user will be directed to the same webserver every time.

The Gravwell provided loadbalancer takes much of the guess work out of configuration and hot failover. While any HTTP loadbalancer that can perform session pinning will work, the native Gravwell load balancer makes configuration easy by directly integrating with the Gravwell datastore so that it can automatically discover available webservers and intelligently failover without the need for a user request to fail.

The Gravwell Loadbalancer is available as a Debian, Redhat, or shell installer; we also publish a docker container in the Docker Hub.

The load balancer configuration is named "loadbalancer.conf" and is typically located in the Gravwell "etc" directory.

Here is an example "loadbalancer.conf" configuration which is serving HTTP requests on port 8080 and communicating with the datastore without a TLS conection:

```
[Global]
        Disable-HTTP-Redirector=true
        Insecure-Disable-HTTPS=true
        Web-Port=8080
        Update-Interval=10
        Session-Timeout=10
        Log-Dir=/opt/gravwell/log/loadbalancer
        Log-Level=info
        Enable-Access-Log=true

        Control-Secret=ControlSecrets
        Datastore=172.19.0.2
        Datastore-Insecure-Disable-TLS=true
        #Datastore-Insecure-Skip-TLS-Verify=true
```

In this case, the loadbalancer will probe the datastore and automatically discover available webservers. As webservers come online or go offline the load balancer will dynamically add and remove the webservers from its available pool.

However, you may wish to manually set some specific webservers that should be considered always online. This does not mean that these webservers will always be used, just that the load balancer will assume they are always supposed to be available. Even with "override" webservers the loadbalancer will route users to an available webserver.

Here is an example configuration with two webservers that have a custom configuration:

```
[Global]
        Disable-HTTP-Redirector=false
        Insecure-Disable-HTTPS=false
        Web-Port=443
        Update-Interval=10
        Session-Timeout=10
        Log-Dir=/opt/gravwell/log/loadbalancer
        Log-Level=info
        Enable-Access-Log=true
```

```
        Control-Secret=ControlSecrets
        Datastore=172.19.0.2
        #Datastore-Insecure-Disable-TLS=true
        Datastore-Insecure-Skip-TLS-Verify=true

# Example setting up an override webserver that may not be in the datastore
[Overrides "example1"]
        Webserver=172.19.0.100
        Insecure-Disable-HTTPS=true

[Overrides "example2"]
        Webserver=172.19.0.101
        Insecure-Disable-HTTPS=true
```

6.3.1 Using Traefik

The Traefik load-balancing proxy has been shown to work well with Gravwell. The following is a sample configuration which load-balances between two Gravwell webservers at 10.0.0.1 and 10.0.0.2:

```
defaultEntryPoints = ["http", "https"]

[file]

[entryPoints]
    [entryPoints.http]
    address = ":80"
    [entryPoints.https]
    address = ":443"
        [entryPoints.https.tls]
            [[entryPoints.https.tls.certificates]]
            certFile = "traefik.crt"
            keyFile = "traefik.key"

[frontends]
    [frontends.frontend1]
        backend = "backend1"
    [frontends.frontend1.headers]
        SSLRedirect = true
        SSLTemporaryRedirect = true

[backends]
    [backends.backend1]
        [backends.backend1.loadbalancer.stickiness]
        [backends.backend1.servers.server1]
            url="https://10.0.0.1"
        [backends.backend1.servers.server2]
            url="https://10.0.0.2"
```

Chapter 7

Ingesters

Ingesters are programs which collect entries from some data source and send them to one or more indexers. The ingesters described in this chapter are all intended to run continuously, receiving newly-generated data and packaging it for Gravwell. If you have pre-existing data (archive files on disk, PCAP files, data in Splunk), see Chapter 11 for information on how best to import it.

7.1 Dealing with Timestamps

All ingesters attach a timestamp to each entry sent to an indexer. Most ingesters extract timestamps directly from the data being ingested; some, such as Kinesis and Pub/Sub, can use a timestamp obtained directly from the upstream data source. When an ingester cannot extract a timestamp, the current time will be applied to the entry.

To find a timestamp in the source data, the ingester will try a list of possible timestamp formats attempting to find a match in the data. It will always try to re-apply the most recent successful format first. For example, if an entry has a timestamp `02 Jan 06 15:04 MST`, the ingester will attempt to parse the *next* entry with the same timestamp format. If it does not match, then the ingester will attempt all other timestamp formats.

There are several ways to change the behavior of how timestamps are parsed, detailed in the next section. Additionally, fully custom timestamp formats can be provided in some ingesters.

7.1.1 Time Zones

Dealing with time zones can be one of the most challenging and frustrating aspects of ingestion. If a log's timestamp includes an explicit UTC offset (e.g. "-0700"), things are relatively easy, but many log formats do not include any time zone information at all! Sometimes, the system generating the log entry is in a local time zone, while the Gravwell ingester's system is set to UTC, or vice versa.

If you believe you have configured your ingester properly, but you're not seeing any data in a query, try expanding your query timeframe to include the future using the "Date Range" timeframe selection: just set the End Date to some time tomorrow. If the Gravwell ingest system is set to a US time zone, but the logs are in UTC time with no offset included, the incoming data will be ingested in the "future".

The `Timezone-Override` parameter (described below) is the surest way to fix time zone problems. If your data has a UTC timestamp but the system clock is set to another time zone, set `Timezone-Override="Etc/UTC"`. If your data is in US Eastern time, but the system clock is set to UTC, set `Timezone-Override="America/New_York"`, and so on.

7.1.2 Time Parsing Overrides

Most ingesters attempt to apply a timestamp to each entry by extracting a timestamp from the data. There are several options which can be applied to each data consumer for fine-tuning of this timestamp extraction:

- `Ignore-Timestamps` (boolean): setting `Ignore-Timestamps=true` will make the ingester apply the current time to each entry rather than attempting to extract a timestamp. This can be the only option for ingesting data when you have extremely incoherent incoming data.
- `Assume-Local-Timezone` (boolean): By default, if a timestamp does not include a time zone the ingester will assume it is a UTC timestamp. Setting `Assume-Local-Timezone=true` will make the ingester instead assume whatever the local computer's timezone is. This is mutually exclusive with the `Timezone-Override` option.
- `Timezone-Override` (string): Setting `Timezone-Override` tells the ingester that timestamps which don't include a timezone should be parsed in the specified timezone. Thus `Timezone-Override=US/Pacific` would tell the ingester to treat incoming timestamps as if they were in US Pacific time. Mutually exclusive with `Assume-Local-Timezone`.
- `Timestamp-Format-Override` (string): This parameter tells the ingester to look for a specific timestamp format in the data, e.g. `Timestamp-Format-Override="RFC822"`. Refer to the timegrinder documentation for a full list of possible overrides, with examples.

7.1.3 Ingester Custom Time Formats

Many ingesters can support the inclusion of custom time formats that can extend the capability of the Gravwell timegrinder time resolution system. Gravwell's timestamp extraction engine has a wide array of timestamp formats that it can automatically identify and resolve. However, in the real world with real developers there is no telling what time format a system may decide to use. That is why we enable users to specify custom time formats for inclusion in the timegrinder system.

Custom time formats are a fallback when the usual timestamp extraction fails; refer to section 7.1 for more general information on timestamp extraction.

Defining a Custom Format

A custom format requires three items to function:

- Name
- Regular Expression
- Format

The given name for a custom time format must be unique across other custom time formats and the included timegrinder formats. For a complete up-to-date listing of included time formats and their names, refer to the timegrinder documentation[1].

Custom time formats are declared in the configuration files for supported ingesters by specifying a named `TimeFormat` block. Here is an example format named "foo" which handles timestamps that are delimited using underscores:

```
[TimeFormat "foo"]
Format="2006_01_02_15_04_05"
Regex=`\d{4}_\d{1,2}_\d{1,2}_\d{1,2}_\d{1,2}_\d{1,2}`
```

This format would properly handle the timestamps in the following logs:

```
2021_02_05_09_00_00 and my id is 1
2021_02_05_09_00_00 and my id is 2
```

[1] https://pkg.go.dev/github.com/gravwell/gravwell/v3/timegrinder#pkg-constants

```
2021_02_05_09_00_00 and my id is 3
2021_02_05_09_00_00 and my id is 4
2021_02_05_09_00_00 and my id is 5
2021_02_05_09_00_00 and my id is 6
```

Here is another format that handles logs with only a timestamp:

```
[TimeFormat "foo2"]
Format="15^04^05"
Regex=`\d{1,2}\^\d{1,2}\^\d{1,2}`
```

This format would handle the following logs, appropriately applying the current date to each extracted timestamp:

```
09^00^00 and my id is 1
09^00^00 and my id is 2
09^00^00 and my id is 3
09^00^00 and my id is 4
09^00^00 and my id is 5
09^00^00 and my id is 6
```

These custom timestamp definitions will be automatically added to the list used for timestamp extraction, but the name can also be explicitly specified in the `Timestamp-Format-Override` parameter. For example, we can force the timestamp format to our custom format using `Timestamp-Format-Override="foo"`.

Time Formats

The `Format` component uses the Go standard time format specification[2]. In short, to define a format, specify the date 'Mon Jan 2 15:04:05 MST 2006' using whatever format you choose.

Time formats can omit the date component. When the custom format system identifies that a custom time format does not include a date component, it will automatically update the extracted timestamp's date to the current day.

Time Zones

All custom time formats will attempt to operate in UTC unless otherwise indicated using the `Format` directive. This means that if you have a time format without a date component you must pay special attention to the timezone. If an application emits a timestamp of "12:00:00" in MST and there is no timezone component or timezone overrides, timegrinder will interpret the timestamp as UTC and the extracted date will be 7 hours in the past.

If your timestamp does contain a timezone you must include that in your `Format` directive so that the timegrinder system knows to interpret the timestamp in the correct time zone. For example here is the previously described "foo" custom format but with a timezone component:

```
[TimeFormat "foo"]
Format="2006_01_02_15_04_05_MST"
Regex=`\d{4}_\d{1,2}_\d{1,2}_\d{1,2}_\d{1,2}_\d{1,2}_\S+`
```

This example will properly handle timestamps in their respective time zones and apply the correct timestamp on extraction.

Example

Here is an example File Follower configuration which adds two custom time formats:

[2] https://golang.org/pkg/time/#pkg-constants

```
[Global]
Ingester-UUID="463c1889-2954-40a0-a3b4-705ea66459f6"
Ingest-Secret = IngestSecrets
Connection-Timeout = 0
Pipe-Backend-Target=/opt/gravwell/comms/pipe #a named pipe connection, this should be used when ingeste
State-Store-Location=/opt/gravwell/etc/file_follow.state
Log-Level=INFO #options are OFF INFO WARN ERROR
Log-File=/opt/gravwell/log/file_follow.log
Max-Files-Watched=64 # Maximum number of files to watch before rotating out old ones, this can be bumpe

#basic default logger, all entries will go to the default tag
#no Tag-Name means use the default tag
[Follower "auth"]
Base-Directory="/tmp/logs/"
File-Filter="*.log" #we are looking for all authorization log files
Tag-Name=test
Assume-Local-Timezone=true #Default for assume localtime is false

[TimeFormat "foo"]
Format="2006_01_02_15_04_05"
Regex=`\d{4}_\d{1,2}_\d{1,2}_\d{1,2}_\d{1,2}_\d{1,2}`

[TimeFormat "foo2"]
Format="15!04!05"
Regex=`\d{1,2}!\d{1,2}!\d{1,2}`
```

The file follower will handle timestamps that are specified as "2021_02_14_12_33_52" and "15!05!22" properly due to the additional custom time formats.

7.2 Configuration

Each ingester uses its own configuration file, typically stored in `/opt/gravwell/etc/`. The config file consists of two sections:

- A `[Global]` section, containing configuration options that apply to the ingester in general.
- One or more named data sources which describe:
  - Where to get data
  - How to process that data into entries

Here is a sample configuration as used by the packet capture ingester:

```
[Global]
Ingest-Secret = IngestSecrets
Pipe-Backend-target=/opt/gravwell/comms/pipe
Log-Level=INFO #options are OFF INFO WARN ERROR
Ingest-Cache-Path=/opt/gravwell/cache/network_capture

[Sniffer "spy1"]
    Interface="p1p1" #sniffing from interface p1p1
    Tag-Name="pcap"
    Snap-Len=0xffff  #maximum capture size
    BPF-Filter="not port 4023"
    Promisc=true
```

Note that it specifies the `[Global]` section and a single `[Sniffer]` section. The `[Sniffer]` section describes where to get data (from interface p1p1), which tag to apply to the entries, and some other options. The `[Global]` section mostly describes how the ingester should connect to an indexer (in this case via Unix pipe) and how to authenticate, but it also sets up a cache and defines the level of logging to output.

The following global configuration options are supported by all ingesters:

- `Ingest-Secret`: the shared secret used to authenticate to an indexer. Must match indexer's Ingest-Auth setting.
- `Insecure-Skip-TLS-Verify`: If set to true, self-signed or otherwise invalid TLS certificates sent by an indexer will be accepted.
- `Ingest-Cache-Path`: A directory which will be used to cache data in case of a network error.
- `Max-Ingest-Cache`: Maximum amount of data to cache in MB.
- `Cache-Depth`: The maximum number of entries to store in memory before writing to disk.
- `Cache-Mode`: Cache mode determines if the cache is always enabled, or only used when indexer connections fail.
- `Log-Level`: A string log level. OFF, INFO, WARN, and ERROR are valid selections.
- `Log-File`: The location to which logs should be written
- `Source-Override`: An IP address (ipv4 or ipv6) which should be put into the "Source" field of entries from this ingester.

NOTE: The `Ingest-Secret` field `must` be set, but all other options can be left blank; the defaults are sensible.

The Global section of the Ingester configuration file must also specify at least one way to connect to an indexer. Ingesters can communicate with an indexer in several ways:

- **Unix pipes**: Gravwell indexers typically put a Unix pipe file at `/opt/gravwell/comms/pipe`. Ingesters on the same system can connect to the pipe for better performance.
  - `Pipe-Backend-target=/opt/gravwell/comms/pipe`
- **Cleartext TCP**: Gravwell indexers can accept unencrypted ingester traffic, typically on port 4023. This should only be used on trusted, internal networks.
  - `Cleartext-Backend-target=127.0.0.1:4023`
- **Encrypted TCP**: Gravwell indexers, if configured with an X509 certificate/key pair, can accept incoming encrypted connections from ingesters. This is the most secure option.
  - `Encrypted-Backend-target=127.1.1.1:4023`

The non-Global configuration options, which define data sources, are unique to each ingester and are described in the individual ingester sections of this document.

7.2.1 Timestamp Format Overrides (Optional)

Data values may contain multiple timestamps, which can cause some confusion when attempting to derive timestamps out of the data. Normally, the Listeners will grab the left most timestamp that can be derived, but it may be desirable to only look for a timestamp in a very specific format. Many ingesters support the ability to specify the exact timestamp format that the ingester via the `Timestamp-Format-Override` parameter. Many timestamp formats are available such as AnsiC, Unix, Ruby, RFC822, RFC822Z, RFC3339, RFC3339Nano, etc. Visit the ingester documentation page for a full list[3].

To force the Listener to only look for timestamps that match the RFC3339 specification add `Timestamp-Format-Override=RFC3339` to the Listener config.

[3] https://docs.gravwell.io/#!ingesters/ingesters.md

7.3 Preprocessors

Sometimes, ingested data needs some additional massaging before being sent to the indexer. Maybe it's JSON data sent over syslog and you wish to strip out the syslog headers, or you're getting gzip-compressed data from an Apache Kafka stream. Maybe you'd like to be able to route entries to different tags based on the contents of the entries. Ingest preprocessors make this possible by inserting one or more processing steps before the entry is sent up to the indexer.

Before those entries are sent to a Gravwell indexer, they may optionally be passed through an arbitrary number of preprocessors. Each preprocessor will have the opportunity to modify the entries. The preprocessors will always be applied in the same order, meaning you could e.g. uncompress the entry's data, then modify the entry tag based on the uncompressed data.

Preprocessors are configured in the ingester's config file using the `Preprocessor` configuration stanza. Each `Preprocessor` stanza must declare the preprocessor module in use via the `Type` configuration parameter, followed by the preprocessor's specific configuration parameters. Consider the following example for the Simple Relay ingester:

```
[Global]
Ingester-UUID="e985bc57-8da7-4bd9-aaeb-cc8c7d489b42"
Ingest-Secret = IngestSecrets
Connection-Timeout = 0
Insecure-Skip-TLS-Verify=true
Cleartext-Backend-target=127.0.0.1:4023 #example of adding a cleartext connection
Log-Level=INFO

[Listener "default"]
    Bind-String="0.0.0.0:7777" #we are binding to all interfaces, with TCP implied
    Tag-Name=default
    Preprocessor=timestamp

[Listener "syslog"]
    Bind-String="0.0.0.0:601" # TCP syslog
    Tag-Name=syslog

[preprocessor "timestamp"]
    Type = regextimestamp
    Regex ="(?P<badtimestamp>.+) MSG (?P<goodtimestamp>.+) END"
    TS-Match-Name=goodtimestamp
    Timezone-Override=US/Pacific
```

This configuration defines two data consumers (Simple Relay calls them "Listeners") named "default" and "syslog". It also defines a preprocessor named "timestamp". Note how the "default" listener includes the option `Preprocessor=timestamp`. This specifies that entries coming from *that* listener on port 7777 should be sent to the "timestamp" preprocessor. Because the "syslog" listener does not set any `Preprocessor` option, entries coming in on port 601 will *not* go through any preprocessors.

The full list of preprocessors, along with complete configuration options for each, is maintained in the Gravwell online documentation.[4] The following is a brief description of some of the more important preprocessors:

- **gzip** - Uncompresses compressed entries.
- **Regex router** - Sets entry tag based on the contents of the entries, using regular expressions to match and extract the contents.
- **Regex timestamp extraction** - Uses regular expressions to extract and parse particularly complicated timestamps in entries.

[4] https://docs.gravwell.io/#!ingesters/preprocessors/preprocessors.md

- **Forwarding** - Forwards a log stream on to another endpoint in addition to ingesting into Gravwell.
- **Gravwell forwarding** - Forwards logs to an additional set of Gravwell indexers, for data duplication.

7.4 Simple Relay Ingester

Simple Relay is the go-to ingester for text based data sources that can be delivered over plaintext TCP and/or UDP network connections via either IPv4 or IPv6.

Some common use cases for Simple Relay are:

- Remote syslog collection
- Devop log collection over a network
- Bro sensor log collection
- Simple integration with any text source capable of delivering over a network

Simple Relay can ingest data in three different formats, treating each one slightly differently:

- Line-delimited text: each line becomes one entry
- RFC5424/RFC3164 syslog: each syslog message is one entry
- JSON: each JSON entity becomes an entry, with the option to change the tag based on the value of a particular field

The Simple Relay ingester uses the unified global configuration block described earlier. Like most other Gravwell ingesters, Simple Relay supports multiple upstream indexers, TLS, cleartext, and named pipe connections, a local cache, and local logging.

Simple Relay's config file specifies Listener blocks to describe how it can accept incoming data. Listeners support several configuration parameters for specifying protocols, listening interfaces and ports, and fine tuning ingest behavior.

The following is a sample configuration showing all three listener types:

```
[Global]
Ingest-Secret = IngestSecrets
Connection-Timeout = 0
Insecure-Skip-TLS-Verify=false
Cleartext-Backend-target=127.0.0.1:4023 #example of a cleartext connection
Cleartext-Backend-target=127.1.0.1:4023 #example of second cleartext connection
Encrypted-Backend-target=127.1.1.1:4024 #example of encrypted connection
Pipe-Backend-Target=/opt/gravwell/comms/pipe #a named pipe connection
Ingest-Cache-Path=/opt/gravwell/cache/simple_relay # local storage cache when uplinks
↪  fail
Max-Ingest-Cache=1024 # Number of MB to store, local cache will only store 1GB before
↪  stopping
Log-Level=INFO
Log-File=/opt/gravwell/log/simple_relay.log

[Listener "default"]
    Bind-String="0.0.0.0:7777"

[Listener "syslogtcp"]
    Bind-String="tcp://0.0.0.0:601" #standard RFC5424 reliable syslog
    Reader-Type=rfc5424
    Tag-Name=syslog
```

```
    Assume-Local-Timezone=true
    Keep-Priority=true

[JSONListener "loglistener"]
    Extractor:"user.country"
    Tag-Match="Japan":users_jp
    Tag-Match="USA":users_usa
    Default-Tag=users_other
```

7.4.1 Listener Types

As mentioned above, Simple Relay supports three different kinds of listeners: line-delimited, syslog, and JSON.

Line-Delimited Listeners

Line-delimited listeners are defined with a `[Listener "listener-name"]` header block as shown in the sample above. They do not need to specify `Reader-Type`, since line-delimited is the default.

A line-delimited listener will generate a new entry for every time it sees a newline in the stream. The `Tag-Name` parameter can be used to define which tag should be applied to entries; if unspecified, "default" will be used.

The `Bind-String` parameter specifies an IP and port on which the collector should listen for incoming flow records. By default, TCP is used. To use UDP instead, add `udp://`, e.g. `udp://0.0.0.0:514`. Specifying an IP of 0.0.0.0 will listen on all available IP addresses.

Syslog Listeners

RFC5424 Syslog listeners are defined with `[Listener "listener-name"]` header block. The `Reader-Type` option must be set to `rfc5424`.

To enable a listener that expects syslog messages using a reliable TCP connection on port 601:

```
[Listener "syslog"]
    Bind-String=0.0.0.0:601
    Reader-Type=RFC5424
```

To accept syslog messages over stateless UDP via port 514:

```
[Listener "syslog"]
    Bind-String=udp://0.0.0.0:514
    Reader-Type=RFC524
```

RFC5424 reader types also support a parameter named `Keep-Priority` which is set to true by default. A typical syslog message is prepended by a priority identifier, however some users may wish to discard the priority from stored messages. This is accomplished by added `Keep-Priority=false` to an RFC5424 listener. Line-based listeners ignore the `Keep-Priority` parameter.

An example syslog message with a priority attached:

```
<30>Sep 11 17:04:14 router dhcpd[9987]: DHCPREQUEST for 10.10.10.82 from
↪  e8:c7:4f:04:e1:af (Chromecast) via insecure
```

An example listener specification which removes the priority tag from entries:

```
[Listener "syslog"]
    Bind-String=udp://0.0.0.0:514
    Reader-Type=RFC524
    Keep-Priority=false
```

The `Tag-Name` parameter can be used to define which tag should be applied to entries; if unspecified, "default" will be used.

The `Bind-String` parameter specifies an IP and port on which the collector should listen for incoming flow records. By default, TCP is used. To use UDP instead, add `udp://`, e.g. `udp://0.0.0.0:514`. Specifying an IP of 0.0.0.0 will listen on all available IP addresses.

JSON Listeners

The JSON Listener type enables some mild JSON processing at the time of ingest, allowing it to apply a unique tag to an entry based on the value of a field in a JSON entry. For instance, it may be useful to send entries to different tags based on a "country" field *in the JSON*, as in the following sample:

```
{ "user": {"name":"Taka", "country":"Japan"}, "ip": "8.83.94.200" }
```

The JSON Listener requires some unique configuration options:

- `Extractor`: specifies the JSON extraction string, for example "user.country".
- `Tag-Match`: specifies which tag should be applied for a given value extracted by the `Extractor`
- `Default-Tag`: specifies the tag to use when no `Tag-Match` is applicable

Given the example JSON shown above, one possible configuration is shown below:

```
[JSONListener "loglistener"]
    Extractor:"user.country"
    Tag-Match="Japan":users_jp
    Tag-Match="USA":users_usa
    Default-Tag=users_other
```

This configuration will examine the "user" element for a sub-element named "country". It will compare the value of the "country" element to the two Tag-Match values "Japan" and "USA"; if there is a match, it will tag the entry `users_jp` or `users_usa`, respectively. If there is no match, the entry will be tagged `users_other`.

The `Bind-String` parameter specifies an IP and port on which the collector should listen for incoming flow records. By default, TCP is used. To use UDP instead, add `udp://`, e.g. `udp://0.0.0.0:514`. Specifying an IP of 0.0.0.0 will listen on all available IP addresses.

7.4.2 Non-Listener-Specific Configuration Options

The configuration options in this section can be set on any type of Listener.

Bind-String

The `Bind-String` parameter controls which interface and port the listener will bind to. The listener can bind to TCP or UDP ports, specific addresses, and specific ports. IPv4 and IPv6 are supported.

```
#bind to all interfaces on TCP port 7777
Bind-String=0.0.0.0:7777

#bind to all interfaces on UDP port 514
Bind-String=udp://0.0.0.0:514

#bind to port 1234 on link local IPv6 address on interface p1p
Bind-String=[fe80::4ecc:6aff:fef9:48a3%p1p1]:1234

#bind to IPv6 globally routable address on TCP port 901
Bind-String=[2600:1f18:63ef:e802:355f:aede:dbba:2c03]:901
```

Ignore-Timestamps (Optional)

The `Ignore-Timestamps` parameter instructs the listener to apply the current timestamp to entries rather than attempting to extract a timestamp from them. This parameter is useful when reading data where there may not be a timestamp present, or the timestamp is wrong on the originating system due to unreliable system clocks. `Ignore-Timestamps` is false by default.

Assume-Local-Timezone (Optional) and Timezone-Override (Optional)

Most timestamp formats have a timezone attached which indicates an offset to Universal Coordinated Time (UTC). However, some systems do not specify the timezone, leaving it up to the receiver to determine what timezone a log entry may be in. `Assume-Local-Timezone` makes the Listener assume that the timestamp is in the same timezone as the Simple Relay reader when the timezone is omitted. `Timezone-Override` takes a string in the IANA timezone database format (e.g. "America/Chicago") and applies that timezone to timestamps which do not specify a timezone.

`Assume-Local-Timezone` and `Timezone-Override` are mutually exclusive.

Source-Override (Optional)

The `Source-Override` parameter instructs the Listener to ignore the source of the data and apply a hard coded value. It may be desirable to hard-code source values for incoming data as a method to organize and/or group data sources. `Source-Override` values can be IPv4 or IPv6 values.

```
Source-Override=192.168.1.1
Source-Override=127.0.0.1
Source-Override=[fe80::899:b3ff:feb7:2dc6]
```

7.4.3 Hands-On Lab: Simple Relay

In this lab we will stand up a Simple Relay ingester and send some traffic to it manually using netcat. First, launch a Gravwell webserver+indexer container:

```
docker run --rm --net gravnet -p 8080:80 -d --name gravwell gravwell:base
```

Open the web interface by pointing your browser at http://localhost:8080 and log in (“admin”/“changeme”). Then start the ingester container running the Simple Relay ingester:

```
docker run --rm --net gravnet --name ingesters -it \
-e GRAVWELL_CLEARTEXT_TARGETS=gravwell:4023 \
gravwell:ingesters /opt/gravwell/bin/gravwell_simple_relay
```

Note the use of the option `-e GRAVWELL_CLEARTEXT_TARGETS=gravwell:4023`. This sets an environment variable within the container that tells the ingester which indexer it should connect to.

The ingesters container ships with a default Simple Relay configuration which listens for line-delimited entries on port 7777, shown below:

```
[Global]
Log-Level=INFO

[Listener "default"]
    Bind-String="0.0.0.0:7777"
    Ignore-Timestamps=true
    Tag-Name=default
```

Note that this configuration does not define an ingest authentication secret or any indexer connection information. These are instead passed as environment variables to the Docker container.

We’ll stand up a third container in order to send some entries by hand:

```
docker run --net gravnet -it busybox /bin/sh
```

From the shell prompt, we can run netcat and hand-type some entries (in this case "foo" and "bar"), then hit Ctrl-C when done:

```
nc ingesters 7777
foo
bar
```

We can verify that the entries have been ingested by running the search `tag=default` as shown in Figure 7.1.

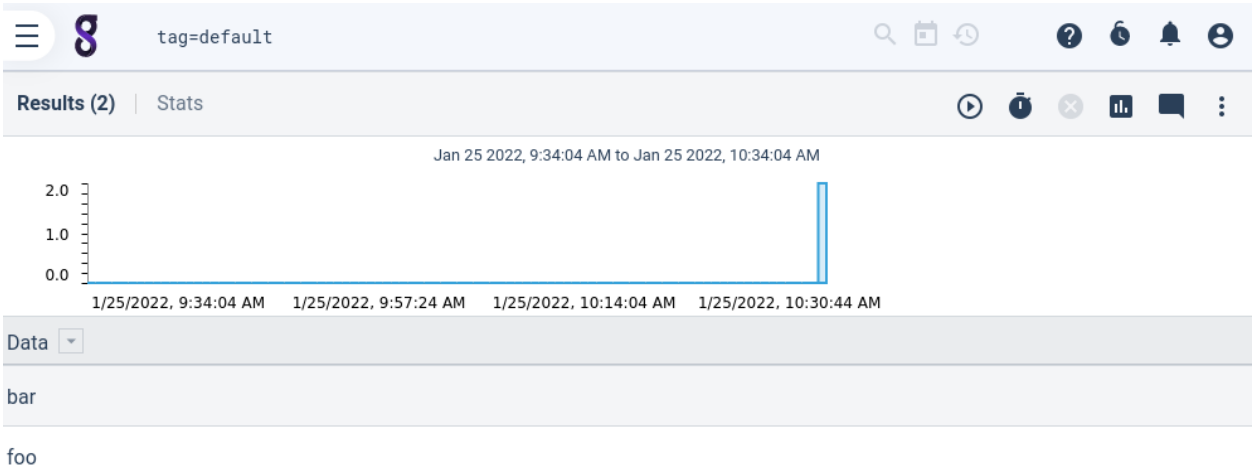


Figure 7.1: Entries from Simple Relay Ingester

Lab Questions

1. How would you convert the listener to a syslog listener?
2. How would you make the Listener listen for incoming UDP rather than TCP?

To clean up after the experiment, simply run:

```
docker rm $(docker ps -a -q)
```

7.5 File Follower Ingester

The File Follower ingester is the best way to ingest files on the local filesystem in situations where those files may be updated on the fly. It ingests each line of the files as a single entry.

The most common use case for File Follower is monitoring a directory containing log files which are actively being updated, such as `/var/log`. It intelligently handles log rotation, detecting when `logfile` has been moved to `logfile.1` and so on. It can be configured to ingest files matching a specific pattern in a directory, optionally recursively descending into the subdirectories of that top-level directory.

Note that if you instead wish to ingest existing/archive files (which will not be updated), the File Follower is not the most efficient option; please see Chapter 11 instead.

The File Follower ingester uses the unified global configuration block described in Section 7.2. Like most other Gravwell ingesters, File Follower supports multiple upstream indexers, TLS, cleartext, and named pipe connections, and local logging.

Below is an example configuration for the File Follower ingester, configured to watch several different types of log files in `/var/log` and recursively follow files under `/tmp/incoming`:

```
[Global]
Ingest-Secret = IngestSecrets
Insecure-Skip-TLS-Verify = false
Cleartext-Backend-target=172.20.0.1:4023
Cleartext-Backend-target=172.20.0.2:4023
State-Store-Location=/opt/gravwell/etc/file_follow.state
Log-Level=ERROR
Max-Files-Watched=64

[Follower "syslog"]
        Base-Directory="/var/log/"
        File-Filter="syslog,syslog.[0-9]"
        Tag-Name=syslog
        Assume-Local-Timezone=true
[Follower "external"]
        Base-Directory="/tmp/incoming"
        Recursive=true
        File-Filter="*.log"
        Tag-Name=external
        Timezone-Override="America/Los_Angeles"
```

In this example, the syslog follower reads `/var/log/syslog` and its rotations, ingesting lines to the syslog tag and assuming dates to be in the local timezone. The external follower reads all files ending in .log from the directory `/tmp/incoming`, descending recursively into directories. It parses timestamps as though they were in the Pacific time zone. This follower illustrates a configuration that would be useful if, for example, several servers on the US west coast periodically uploaded their log files to this system.

The configuration parameters used above are explained in greater detail in the following sections.

7.5.1 Additional Global Parameters

The File Follower ingester defines several additional parameters for the `[Global]` section. These parameters apply to *all* Followers defined in the config file.

Max-Files-Watched

The `Max-Files-Watched` parameter prevents the File Follower from maintaining too many open file descriptors. If `Max-Files-Watched=64` is specified, the File Follower will actively watch up to 64 log files. When a new

file is created, the File Follower will stop actively watching the oldest existing file in order to watch the new one. However, if the old file is later updated, it will return to the top of the queue.

We recommend leaving this setting at 64 in most cases; configuring the limit too high can run into limits set by the kernel.

7.5.2 Follower Configuration Parameters

The File Follower configuration file contains one or more "Follower" directives:

```
[Follower "syslog"]
        Base-Directory="/var/log/"
        File-Filter="syslog,syslog.[0-9]"
        Tag-Name=syslog
```

Each follower specifies at minimum a base directory and a filename filtering pattern. This section describes possible configuration parameters which can be set per follower.

Base-Directory

The `Base-Directory` parameter specifies the directory which will contain the files to be ingested. It should be an absolute path and contain no wildcards.

File-Filter

The `File-Filter` parameter defines the filenames which should be ingested. It can be as simple as a single file name:

```
File-Filter="foo.log"
```

Or it can contain multiple patterns:

```
File-Filter="kern*.log,kern*.log.[0-9]"
```

which will match any filenames beginning with "kern" and ending with ".log", or beginning with "kern" and ending with ".log.0" through ".log.9".

The full matching syntax, as defined in https://golang.org/pkg/path/filepath/#Match, is:

```
pattern:
    { term }
term:
    '*'         matches any sequence of non-Separator characters
    '?'         matches any single non-Separator character
    '[' [ '^' ] { character-range } ']'
                character class (must be non-empty)
    c           matches character c (c != '*', '?', '\\', '[')
    '\\' c      matches character c

character-range:
    c           matches character c (c != '\\', '-', ']')
    '\\' c      matches character c
    lo '-' hi   matches character c for lo <= c <= hi
```

Tag-Name (Optional)

The `Tag-Name` parameter specifies the tag to apply to entries ingested by this follower. If unset, the "default" tag is used.

Ignore-Timestamps (Optional)

The `Ignore-Timestamps` parameter indicates that the follower should not attempt to extract a timestamp from each line of the file, but rather tag each line with the current time.

Assume-Local-Timezone (Optional)

`Assume-Local-Timezone` is a boolean setting which directs the ingester to parse timestamps which lack timezone specifications as though they were in the local time zone rather than the default UTC.

`Assume-Local-Timezone` and `Timezone-Override` are mutually exclusive.

Timezone-Override (Optional)

The `Timezone-Override` parameter directs the ingester to parse timestamps which lack timezone specifications as though they were in the given time zone rather than the default UTC.

The timezone should be specified in IANA database string format as defined at https://en.wikipedia.org/wiki/List_of_tz_databa
for example, US Central Time should be specified as follows:

```
Timezone-Override="America/Chicago"
```

`Assume-Local-Timezone` and `Timezone-Override` are mutually exclusive.

`Timezone-Override="Local"` is functionally equivalent to `Assume-Local-Timezone=true`

Recursive (Optional)

The `Recursive` parameter directs the File Follower to ingest all files matching the `File-Filter` *recursively* under the `Base-Directory`.

By default, the ingester will only ingest those files matching the `File-Filter` under the top level of the `Base-Directory`; the following would ingest `/tmp/incoming/foo.log` but not `/tmp/incoming/system1/foo.log`:

```
Base-Directory="/tmp/incoming"
File-Filter="foo.log"
```

By setting `Recursive=true`, the configuration will ingest *any* file named `foo.log` at any directory depth under `/tmp/incoming`.

Ignore-Line-Prefix (Optional)

The ingester will drop (not ingest) any lines beginning with the string passed to `Ignore-Line-Prefix`. This is useful when ingesting log files which contain comments, such as Bro logs. The `Ignore-Line-Prefix` parameter may be specified multiple times. The following indicates that lines beginning with # or // should not be ingested:

```
Ignore-Line-Prefix="#"
Ignore-Line-Prefix="//"
```

Each follower can specify a unique timestamp format using the `Timestamp-Format-Override` parameter.

Timestamp-Delimited (Optional)

The `Timestamp-Delimited` parameter is a boolean specifying that each occurrence of a time stamp should be considered the start of a new entry. This is useful when log entries may span multiple lines. When specifying `Timestamp-Delimited`, the `Timestamp-Format-Override` parameter must also be set. Let's look at a file snippet and explore how timestamp delimiting would affect entry extraction. If we assume the following snippet:

```
2012-11-01T22:08:41+00:00 Line 1 of the first entry
Line 2 of the first entry
2012-11-01T22:08:43+00:00 Line 1 of the second entry
Line 2 of the second entry
```

The entries extracted would be:

Entry 1:

```
2012-11-01T22:08:41+00:00 Line 1 of the first entry
Line 2 of the first entry
```

Entry 2:

```
2012-11-01T22:08:43+00:00 Line 1 of the second entry
Line 2 of the second entry
```

7.5.3 Hands-On Lab: File Follower

In this lab, we will use the File Follower to ingest Linux system logs. First, launch a Gravwell webserver+indexer container:

```
docker run --rm --net gravnet -p 8080:80 -d \
--name gravwell gravwell:base
```

Next, we create the ingester container. It needs some additional configuration, so we use `docker create` rather than `docker start`:

```
docker create --rm --net gravnet --name ingesters -it -e \
GRAVWELL_CLEARTEXT_TARGETS=gravwell:4023 -v /var/log:/var/log \
gravwell:ingesters /opt/gravwell/bin/gravwell_file_follow -v
```

Note the option `-v /var/log:/var/log`. This mounts the host's log directory into the container. In this lab, we will be ingesting the *host's* logs, since Docker containers do not generate many log files.

The File Follower in the container is pre-configured with these Follower definitions:

```
[Follower "auth"]
    Base-Directory="/var/log/"
    File-Filter="auth.log,auth.log.[0-9]"
    Tag-Name=auth
[Follower "kernel"]
    Base-Directory="/var/log"
    File-Filter="dmesg,dmesg.[0-9]"
    Tag-Name=kernel
[Follower "kernel2"]
    Base-Directory="/var/log"
    File-Filter="kern.log,kern.log.[0-9]"
    Tag-Name=kernel
```

There is a problem with the configuration as it stands. Linux log timestamps don't include a time zone, and your system is probably set to the local time zone, so the log files will contain entries with local timestamps. Docker containers default to UTC, though, so the ingester will improperly parse the log timestamps. Luckily, this can be fixed in the config file.

NOTE: If your host computer is set to UTC, you can skip these config modifications and skip straight to starting the ingester.

We'll first copy out the config file from the ingester:

```
docker cp ingesters:/opt/gravwell/etc/file_follow.conf /tmp
```

Edit the file and add the following to each `Follower`block, substituting your timezone:

```
Timezone-Override="America/Denver"
```

And push the configuration back to the ingester:

```
docker cp /tmp/file_follow.conf ingesters:/opt/gravwell/etc/file_follow.conf
```

With the configuration fixed, we can start the ingester:

```
docker start ingesters
```

Recall that we passed the -v flag to `gravwell_file_follow` when we created the container; this enables verbose mode, which allows us to see every entry the ingester reads *and* the timestamp it derived from the log entry. Most ingesters support the -v flag. You can view the log output by running the command `docker logs ingesters`; the following is a sample:

```
GOT 2019-03-09T13:12:59-07:00 Mar  9 13:12:59 bombadil kernel:  [7955880.007543] hub
2-1:1.0:  3 ports detected
```

The portion `GOT 2019-03-09T13:12:59-07:00` indicates that the ingester has parsed out a timestamp of March 9, 13:12:59 US Mountain Time from the log entry, which comprises the rest of the line.

Based on the configuration, we can expect to see entries tagged "auth" and "kernel", and a quick search for `tag=auth` should confirm this, as shown in Figure 7.2.

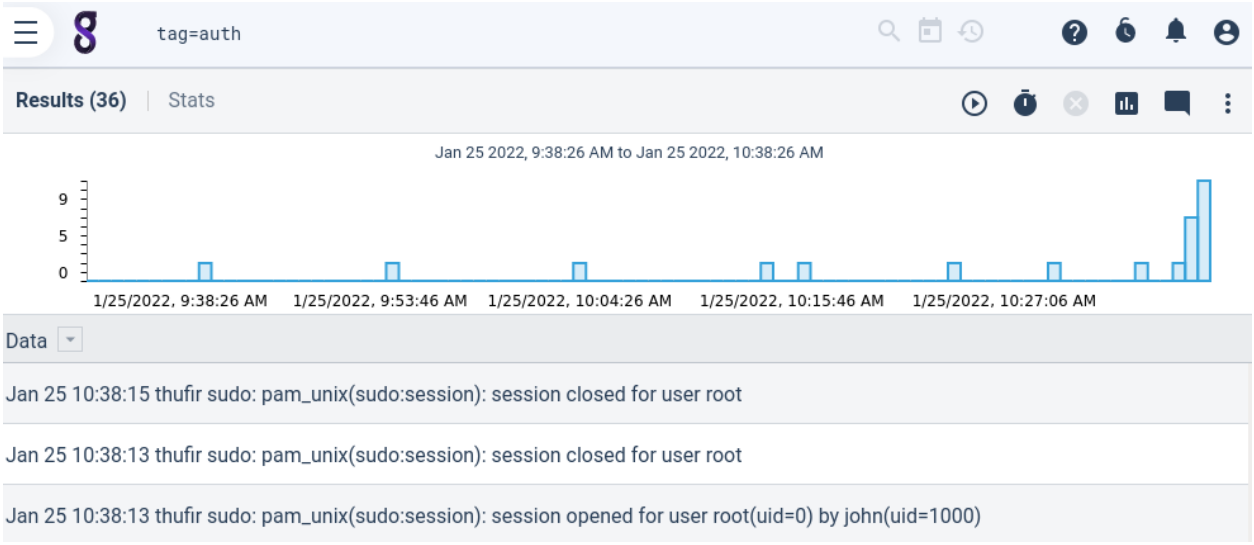


Figure 7.2: Auth logs ingested via File Follower

Lab Questions

1. Suppose we wanted to ingest *all* files ending in `.log` in `/var/log`, including files in subdirectories. Configure a Follower that would accomplish this.

To clean up after the experiment, simply run:

```
docker kill $(docker ps -a -q)
```

7.6 Windows Event Ingester

The Windows event ingester is designed to monitor the Windows event service and ingest raw events into Gravwell. Windows events are formatted in XML and frankly, can get a little unruly; as a result the Windows ingester supports the ability to subscribe to specific event streams and filter based on Provider and EventID directly in the ingester. While Gravwell is typically an "ingest first and ask questions later" kind of company, the Windows logging system produces a tremendous amount of un-useful events that are large and complex to parse. If you are monitoring significant Windows infrastructure, it may not be necessary to ingest 100GB of XML that tells you every time a desktop goes to sleep or wakes up.

The Windows event ingester is a system service and is installed into `%ProgramFiles%\gravwell`. The service is configured via a `config.cfg` file located in the same directory. Configuration is similar to other ingesters and shares the common `[Global]` block which defines how the ingester will authenticate and communicate with indexers. After the `[Global]` section, the config file can define subscriptions for Windows event channels. Each EventChannel can specify a tag, maximum reachback, and filters for the provider, event id, and level. Available configuration parameters for an EventChannel are as follows:

- `Tag-Name`: Specifies the tag the events will go to.
- `Provider`: Filter to control which providers the ingester will send to indexers.
- `EventID`: Controls which event IDs are sent to indexers.
- `Level`: Controls minimum alert level.
- `Max-Reachback`: Controls a maximum age of entries the ingester will read on first start.

For example, let's look at an event channel configuration which subscribes to the "security" channel:

```
[EventChannel "security"]
    Tag-Name=windows
    Channel=Security
```

This configuration specifies that the ingester should send all events on the security channel to the tag windows. Because we haven't specified any filters or a `Max-Reachback`, the ingester will grab all historical events that are available from the `Security` channel and send every one of them to the indexer. It then begins streaming new events as they come in.

Let's next examine a configuration that applies some filters to control what we ingest:

```
[EventChannel "sysmon"]
    Tag-Name=sysmon
    Channel=Microsoft-Windows-Sysmon/Operational
    Max-Reachback=24h
    Level=4
    EventID=4000-5000
```

This configuration reads entries from the `Microsoft-Windows-Sysmon/Operational` channel, but only if the Level is 4 and the EventID is between 4000 and 5000. When the Ingester first starts it will also retrieve entries from the Windows event store that are up to 24 hours old. Tailoring the configuration can help narrow in on event sources and event IDs that actually have relevance to your mission, whether it be compliance, security, or performance monitoring.

Let's take a look at a windows log entry that has been formatted so that we can see some of its structure:

```
<?xml version="1.0" encoding="UTF-8"?>
<Event xmlns="http://schemas.microsoft.com/win/2004/08/events/event">
   <System>
      <Provider Name="Microsoft-Windows-Kernel-Power"
Guid="{331c3b3a-2005-44c2-ac5e-77220c37d6b4}" />
      <EventID>107</EventID>
```

```
      <Version>1</Version>
      <Level>4</Level>
      <Task>102</Task>
      <Opcode>0</Opcode>
      <Keywords>0x8000000000000444</Keywords>
      <TimeCreated SystemTime="2019-03-23T21:15:39.4065468Z"/>
      <EventRecordID>2243</EventRecordID>
      <Correlation />
      <Execution ProcessID="4" ThreadID="9996"/>
      <Channel>System</Channel>
      <Computer>rebecca-PC</Computer>
      <Security />
   </System>
   <EventData>
      <Data Name="TargetState">4</Data>
      <Data Name="EffectiveState">5</Data>
      <Data Name="WakeFromState">4</Data>
      <Data Name="WakeRequesterTypeAc">2</Data>
      <Data Name="WakeRequesterTypeDc">0</Data>
   </EventData>
</Event>
```

The log is large and unruly, there is a lot of repetition. Fortunately the `winlog` search module can help simplify working with windows XML logs. Let's start by executing a query that extracts the Provider and EventID. We will then filter the extracted values to only look at the `Microsoft-Windows-Sysmon` provider and EventID 1. The winlog search module supports several shortcuts that can make it easier to extract data from the Windows XML structures. The shortcuts allow us to specify most of the system members directly by name, any other arguments which do not specify a system field member is assumed to be a data field member. Refer to the winlog online documentation for a full list of extractions.[5]

For example, if we want the Provider and EventID fields, we can execute the query `winlog Provider EventID`; the winlog module knows where those fields are located within the XML document. If we wanted the data field named "TargetState", we can just pass the argument `TargetState`. The `winlog` module figures out that it's not a known system field and assumes it is a data name field. Given the example entry above, if we executed the following query:

```
tag=windows winlog Provider EventID WakeFromState
```

The following Enumerated values would be extracted:

Name	**Value**
Provider	Microsoft-Windows-Kernel-Power
EventID	107
WakeFromState	4

The winlog module supports inline filtering (which is covered in depth in later chapters), so we can pass equality statements to each argument to filter down to specific events. For example, if we wanted to only look at events from the `Microsoft-Windows-Sysmon` provider we would execute the following query:

```
tag=windows winlog Provider=="Microsoft-Windows-Sysmon"
```

[5] https://docs.gravwell.io/#!search/winlog/winlog.md

7.6.1 Hands-on Lab: Windows logs

Windows logs are normally XML, which is a large and often complex format. Gravwell provides the `winlog` search module that can help with extracting useful data from Windows logs. For this lab we will examine some Windows logs and extract some basic data so that we can track process creation using sysmon[6] using the SwiftOnSecurity configuration set[7]. For more information refer to our our Windows events documentation page[8]

We can't run the Windows ingester on Docker, so instead we will just import some Windows logs and poke around a bit. We'll fire up a test container and push in the Windows logs using the `regexFile` ingester. The regexFile ingester is a one-off ingester that can assist in ingesting single files where the entry delimiter is more complex than a simple newline. Windows logs often have spaces and newlines in them, so we are going to use the regexFile ingester to break on XML boundaries. Take note of the `rexp` argument; it's a regular expression that will match the beginning of a Windows log entry.

First, we start the indexer and webserver:

```
docker run --rm -d -p 8080:80 --net gravnet --name test gravwell:base
```

Now we'll push in some data with the reimport ingester:

```
cd ~/gravwell_training/Ingesters/Lab-Winevent
docker run -v $PWD/data:/tmp/data --rm -i --net gravnet \
gravwell:ingesters /opt/gravwell/bin/reimport -rebase-timestamp \
-clear-conns test:4023 -i /tmp/data/winlog.json.gz -import-format json
```

Log into the web GUI (http://localhost:8080) and perform the following queries:

1. Use the winlog search module to get all entries that represent process creation events with the following constraints:
   (a) Provider is `Microsoft-Windows-Sysmon`
   (b) EventID is 1
2. Extract the Computer and CommandLine fields and make a table with those columns

Lab Questions:

1. Do you see any command line processes that look interesting?
2. Are there any large parameters that you could decode?
   (a) How would you do it?

To clean up after the experiment, simply run:

```
docker kill $(docker ps -a -q)
```

[6] https://docs.microsoft.com/en-us/sysinternals/downloads/sysmon
[7] https://raw.githubusercontent.com/SwiftOnSecurity/sysmon-config/master/sysmonconfig-export.xml
[8] https://docs.gravwell.io/#!ingesters/ingesters.md#Windows_Event_Service

7.7 Netflow and IPFIX Ingester

The Netflow ingester acts as a Netflow collector, gathering records created by Netflow exporters and capturing them as Gravwell entries for later analysis. These entries can then be analyzed using the netflow search module. The Netflow ingester currently only accepts traffic via UDP.

A sample configuration is shown below:

```
[Global]
Ingest-Secret = IngestSecrets
Connection-Timeout = 0
Insecure-Skip-TLS-Verify=false
Pipe-Backend-target=/opt/gravwell/comms/pipe #a named pipe connection, this should be
↪  used when ingester is on the same machine as a backend
Log-Level=INFO

[Collector "netflow v5"]
    Bind-String="0.0.0.0:2055" #we are binding to all interfaces
    Tag-Name=netflow

[Collector "ipfix"]
    Tag-Name=ipfix
    Bind-String="0.0.0.0:6343"
    Flow-Type=ipfix
```

This configuration defines two collectors, one for Netflow v5, one for IPFIX. The Netflow collector listens on UDP port 2055 and tags its entries "netflow", while the IPFIX collector listens on UDP port 6343 and tags its entries "ipfix".

Note that the "ipfix" collector type can accept both IPFIX and Netflow v9 data.

7.7.1 Collector Configuration Parameters

Bind-String

The `Bind-String` parameter specifies an IP and UDP port on which the collector should listen for incoming flow records. Specifying an IP of 0.0.0.0 will listen on all available IP addresses.

Tag-Name (Optional)

The `Tag-Name` parameter specifies the tag to apply to entries ingested by this follower. If not specified, the "default" tag is used.

Flow-Type (Optional)

The `Flow-Type` parameter specifies which type of flows the collector should read. The available options are "netflowv5" and "ipfix"; the default is "netflowv5".

7.7.2 Hands-on Lab: Netflow Ingester

In this lab, we will ingest procedurally-generated Netflow records. First, launch a Gravwell webserver+indexer container:

```
docker run --rm --net gravnet -p 8080:80 -d --name gravwell gravwell:base
```

Next, start the ingester container running the netflow ingester:

```
docker run --rm --net gravnet --name ingesters -it \
-e GRAVWELL_CLEARTEXT_TARGETS=gravwell:4023 \
gravwell:ingesters /opt/gravwell/bin/gravwell_netflow_capture
```

The netflow ingester is pre-configured to listen on port 2055 for incoming Netflow v5 records.

Now, we use another Docker container to generate Netflow records and send them to the ingester:

```
docker run -it --net gravnet --rm \
networkstatic/nflow-generator -t ingesters -p 2055
```

The netflow generator will run indefinitely, generating flow records, until killed.

Log into the web GUI (http://localhost:8080). We can now search the netflow data:

```
tag=netflow netflow Bytes Src | stats sum(Bytes) by Src | table Src sum
```

We should see traffic originating from many (randomly-generated) IP addresses, as shown in Figure 7.3.

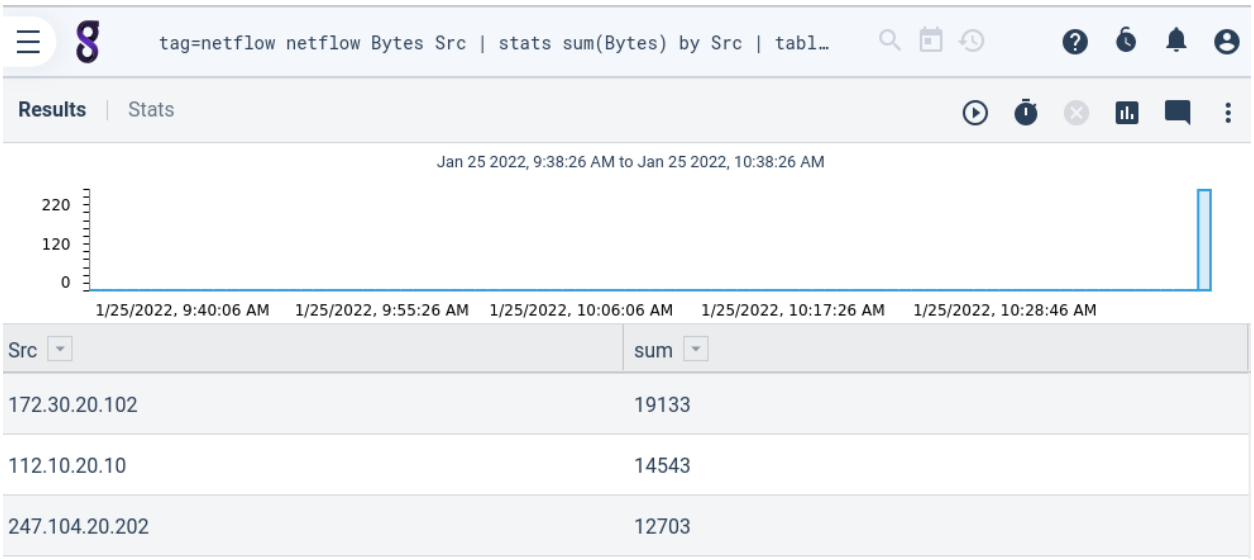


Figure 7.3: Netflow Query Results

To clean up after the experiment, simply run:

```
docker kill $(docker ps -a -q)
```

7.8 Packet Capture Ingester

The packet capture ingester illustrates one of Gravwell's unique strengths: its ability to ingest raw, unprocessed binary data. This ingester listens on one or more network interfaces and ingests every packet it sees in a separate entry. Later, those packets can be parsed using the 'packet' search module.

A sample configuration is shown below:

```
[Global]
Ingest-Secret = IngestSecrets
Connection-Timeout = 100s
Cleartext-Backend-target=192.168.0.1:4023 #example of adding another cleartext connection
Max-Ingest-Cache=32
Ingest-Cache-Path=/opt/gravwell/cache/network_capture
```

```
[Sniffer "spy1"]
    Interface="eth0" #sniffing from interface eth0
    Tag-Name="pcap"  #assigning tag pcap
    Snap-Len=0xffff  #maximum capture size
    BPF-Filter="not port 4023" #do not sniff any traffic on our backend connection
    Promisc=true
```

This configuration defines a single Sniffer, which listens for packets on interface `eth0`, tags them as "pcap", and ships them to an indexer via an unencrypted connection. Note the use of the `BPF-Filter` option; this tells the ingester to ignore packets on port 4023, because the ingester is using port 4023 to transfer entries to the indexer. If this option was not set, every entry sent to the indexer would result in at least one additional entry being generated!

7.8.1 Hands-on Lab: Packet Capture Ingester

This lab will demonstrate ingestion of packets. First, launch a base Gravwell container:

```
docker run --rm --net gravnet -p 8080:80 -d --name gravwell gravwell:base
```

Log into the web GUI (http://localhost:8080) once the container is started.

Next, start the ingester container running the packet capture ingester:

```
docker run --rm --net gravnet --name pcap -d \
-e GRAVWELL_CLEARTEXT_TARGETS=gravwell:4023 \
gravwell:pcap /opt/gravwell/bin/gravwell_network_capture
```

The ingester is pre-configured to capture packets on the eth0 interface. The ingester should start seeing packets soon, even without generating any traffic; this can be verified by searching for `tag=pcap` as shown in Figure 7.4.

Note that the packets are "garbage". This is because the packets were captured as binary blobs, and without specifying how to parse them, Gravwell can only print an ASCII representation of the binary.

We can see which protocols are in use by extracting the source IP, dest IP, and IP protocol from each packet. Results are shown in Figure 7.5.

```
tag=pcap packet ipv4.SrcIP ipv4.DstIP ipv4.Protocol | table
```

The screenshot shows that the majority of packets are currently protocol 17 (UDP).

Next, we'll generate some ping traffic:

```
docker run -it --net gravnet --rm gravwell:base ping pcap
```

We can then run a search which limits results only to protocol 1 (ICMP):

```
tag=pcap packet ipv4.SrcIP ipv4.DstIP ipv4.Protocol==1 | table
```

We can see which protocol is appearing most frequently by running the following query:

```
tag=pcap packet ipv4.Protocol | count by Protocol | chart count by Protocol
```

The results are shown in Figure 7.6, with the chart changed to a pie chart.

To clean up after the experiment, simply run:

```
docker kill $(docker ps -a -q)
```

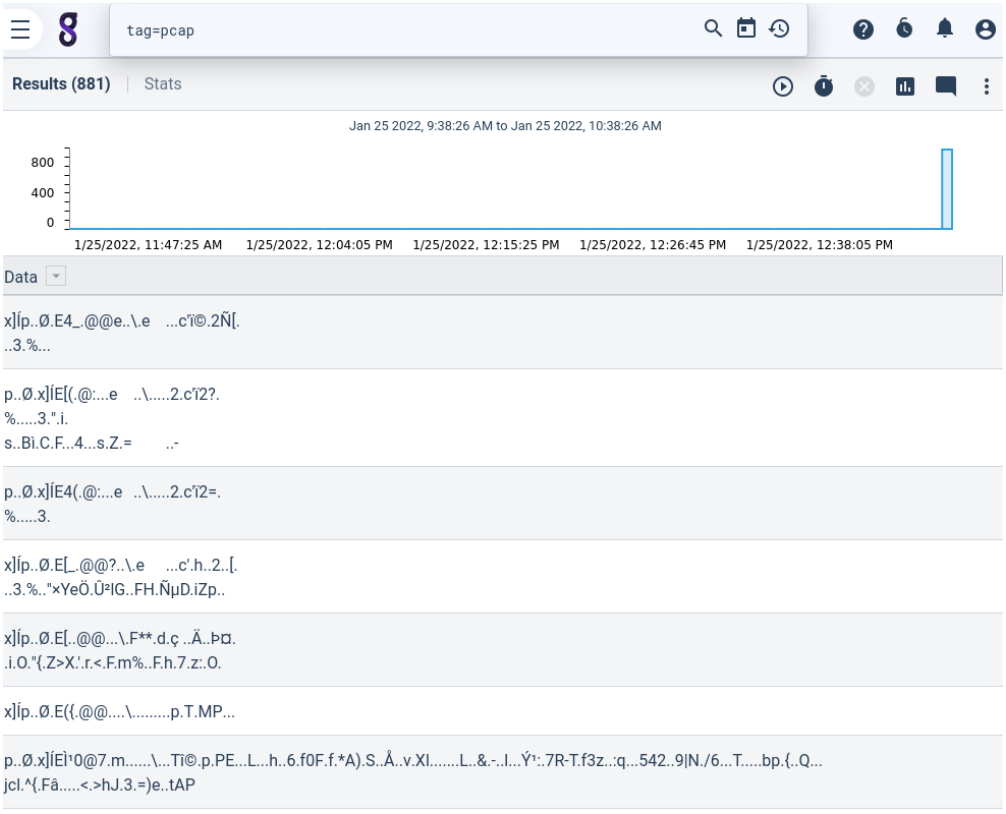


Figure 7.4: Raw packets

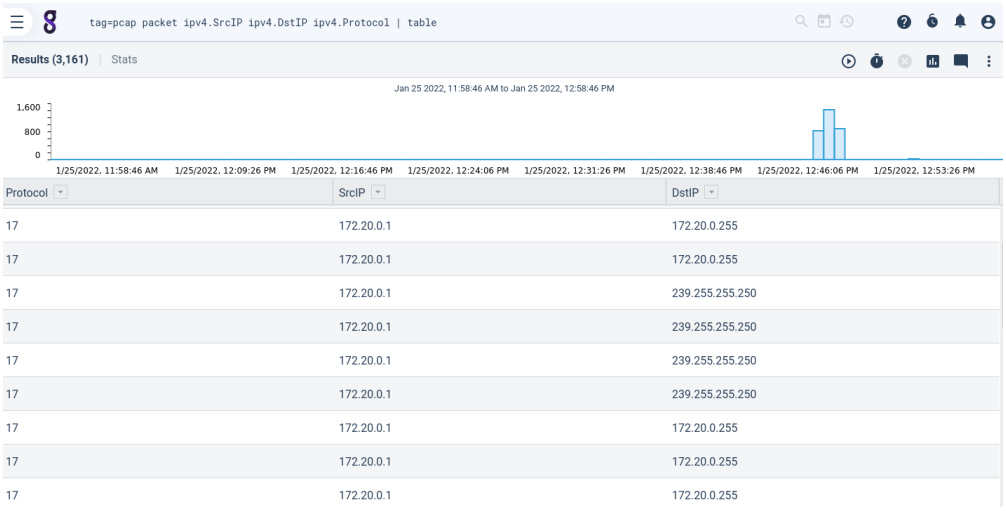


Figure 7.5: Parsing Packets

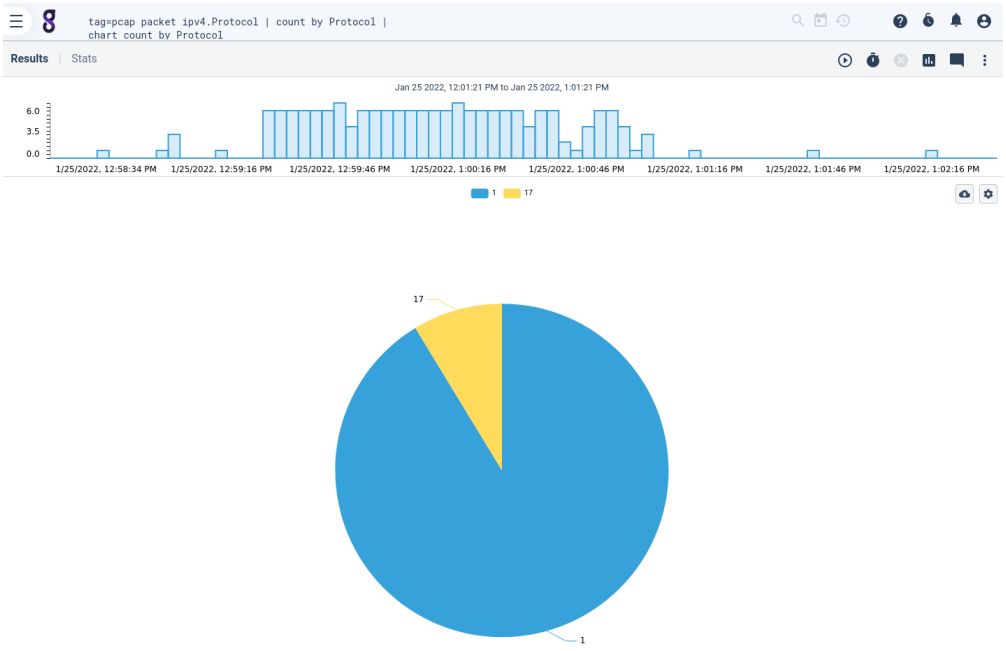


Figure 7.6: Protocol Ratios

7.9 Tag Management / Federation

The Federator is an entry relay: ingesters connect to the Federator and send it entries, then the Federator passes those entries to an indexer. The Federator can act as a trust boundary, securely relaying entries across network segments without exposing ingest secrets or allowing untrusted nodes to send data for disallowed tags. The Federator upstream connections are configured like any other ingester, allowing multiplexing, local caching, encryption, etc. Figure 7.7 shows an example architecture in which multiple Federators are used to gather data from different business units into a central indexer pool.

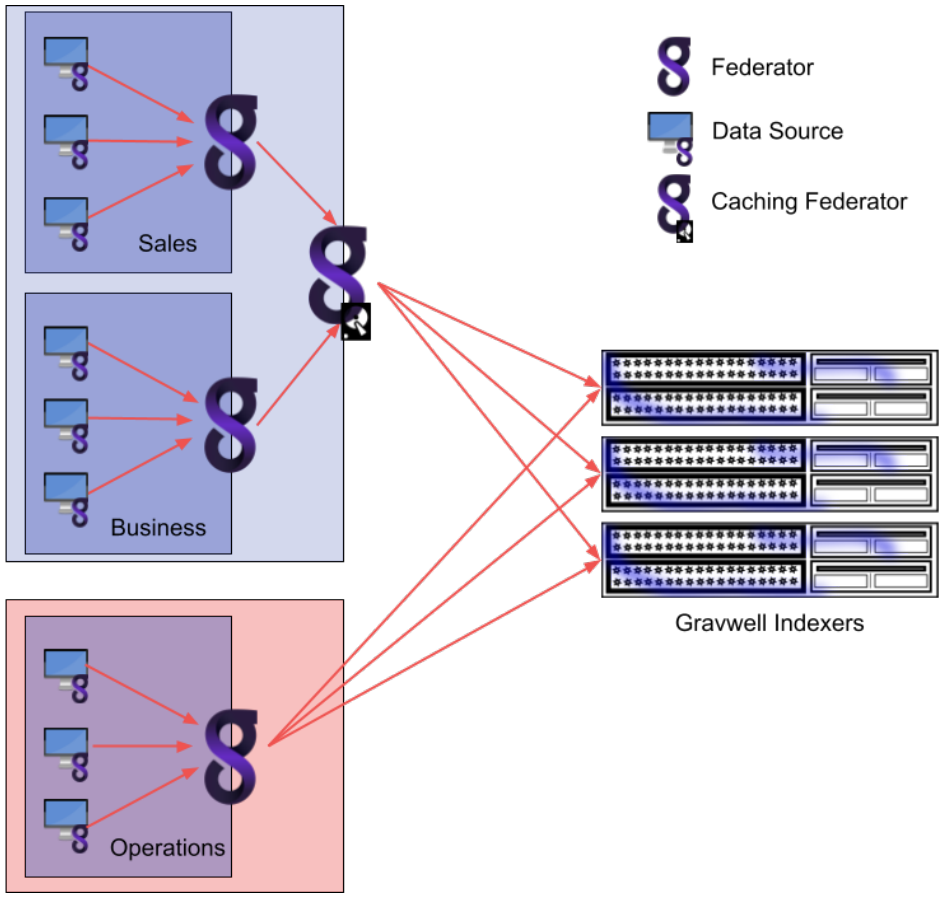


Figure 7.7: A Federated Architecture

The Federator is well-suited for several use cases:

- Ingesting data across geographically diverse regions when there may not be robust connectivity
- Providing an authentication barrier between network segments
- Reducing the number of connections to an indexer
- Controlling the tags and data source group can provide

The Federator can improve security by enforcing two kinds of controls:

- Auth: The Federator can use different Ingest Secrets on different listening ports.
  - Example: Assume two customers which will be uploading entries from ingesters in their own infrastructure to your indexer. By configuring a Federator with a separate listener and unique Ingest Secret for each customer, you can easily revoke one customer's access.

- Tags: The Federator can restrict which tags ingesters are allowed to use.
  - Example: If multiple customers are uploading entries from their own ingesters, a Federator can ensure that Customer A only uploads entries tagged "custA_data" and Customer B only uploads entries tagged "custB_data".

A sample configuration is shown below:

```
[Global]
    Ingest-Secret = SuperSecretUpstreamIndexerSecret
    Connection-Timeout = 0
    Insecure-Skip-TLS-Verify = false
    Encrypted-Backend-target=172.20.232.105:4024
    Encrypted-Backend-target=172.20.232.106:4024
    Ingest-Cache-Path=/opt/gravwell/cache/federator
    Max-Ingest-Cache=1024 #1GB
    Log-Level=INFO

[IngestListener "BusinessOps"]
        Ingest-Secret = CustomBusinessSecret
        Cleartext-Bind = 10.0.0.121:4023
        Tags=windows-*
        Tags=syslog

[IngestListener "DMZ"]
       Ingest-Secret = OtherRandomSecret
       TLS-Bind = 192.168.220.105:4024
       TLS-Certfile = /opt/gravwell/etc/cert.pem
       TLS-Keyfile = /opt/gravwell/etc/key.pem
       Tags=apache
       Tags=nginx
```

This configuration connects to two upstream indexers in a *protected* network segment and provides ingest services on two *untrusted* network segments. Each untrusted ingest point has a unique `Ingest-Secret`, with one serving TLS with a specific certificate and key pair. The configuration file also enables a local cache, making the Federator act as a fault-tolerant buffer between the Gravwell indexers and the untrusted network segments.

7.9.1 Wildcard Tags

When specifying an `IngestListener`, you may wish to accept a wide range of tags instead of just a handful of specific ones. Perhaps you're receiving Windows event logs from hundreds of different machines, and each one includes its hostname in the tag, e.g. "winlog-activedirectory01". The `Tags` field can contain wildcards, so you could match the example given by simply saying `Tags=winlog-*`.

7.9.2 Hands-on Lab: Federation

In this lab, we will configure a Federator with tag restrictions, then attempt to send it entries with both allowed and disallowed tags. First, launch a Gravwell webserver+indexer container:

```
docker run --rm --net gravnet -p 8080:80 -d --name gravwell gravwell:base
```

Log into the web GUI (http://localhost:8080) and keep it open.

Start another container running the Federator:

```
docker run --rm --net gravnet --name federator -it \
-e GRAVWELL_CLEARTEXT_TARGETS=gravwell:4023 \
```

```
gravwell:ingesters /opt/gravwell/bin/gravwell_federator
```

The container we are using has two pre-configured listeners for the Federator:

- `enclaveA`, which listens on port 4001, allows entries tagged "testA", and uses the Ingest Secret "enclaveASecrets"
- `enclaveB`, which listens on port 4002, allows entries tagged "testB", and uses the Ingest Secret "enclaveBSecrets"

With the Federator up and running, we use the generators container to attempt to send some JSON-formatted entries to the Federator. We direct it to connect to the "federator" container on port 4001 and use the secret "enclaveASecrets", which is the correct configuration for the first IngestListener defined in the Federator config. However, note that we add the option `-tag-name json`. This is not one of the allowed tags!

```
docker run --net gravnet --rm -it gravwell:generators \
/jsonGenerator -clear-conns federator:4001 \
-ingest-secret enclaveASecrets -tag-name json
```

The command should fail, with an error message saying it failed to negotiate tags:

```
2020/02/05 18:44:25 ERROR: Timed out waiting for active connection due to All connections
↪  failed Failed to negotiate tags
```

This is expected, because we attempted to send entries tagged "json", which is not allowed! Let's run the generator again, this time specifying the allowed "testA" tag:

```
docker run --net gravnet --rm -it gravwell:generators \
/jsonGenerator -clear-conns federator:4001 \
-ingest-secret enclaveASecrets -tag-name testA
```

We can verify that the entries made it in by running the query `tag=testA` in the Gravwell interface, as shown in Figure 7.8.

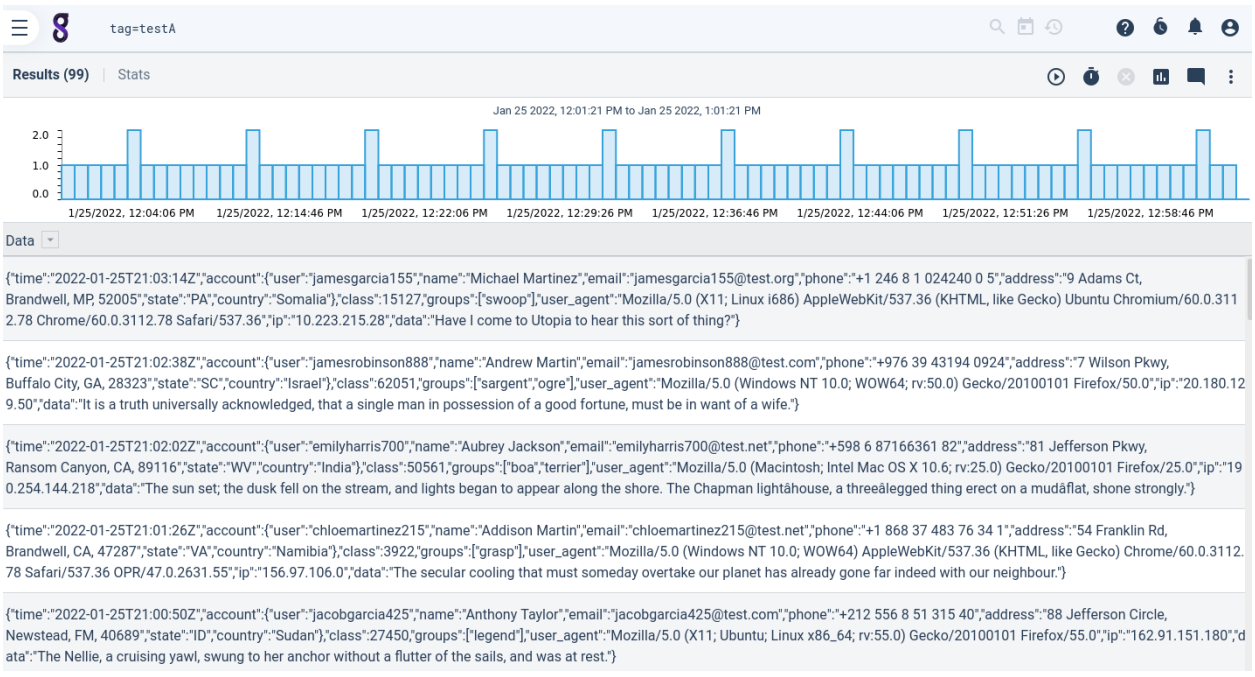


Figure 7.8: Entries Ingested via Federator

We can send entries to the other Federator listener by tweaking the generator command line, specifying port 4002, the secret "enclaveBSecrets", and tag "testB":

```
docker run --net gravnet --rm -it gravwell:generators \
/jsonGenerator -clear-conns federator:4002 \
-ingest-secret enclaveBSecrets -tag-name testB
```

To clean up after the experiment, simply run:

```
docker kill $(docker ps -a -q)
```

7.10 Ingester Caching

Ingester caching allows Gravwell ingesters to continue receiving data and generating entries even when there are no active connections to indexers. Entries are stored in an on-disk cache until indexer communication is reestablished, at which point they are uploaded to the indexer. Additionally, the cache can be configured to always be engaged, allowing the ingester to absorb bursts of data that cannot be sent to an indexer in real time.

All officially-supported ingesters support caching. Caching is controlled by four configuration options in the `[Global]` section of the ingester configuration file:

- `Ingest-Cache-Path`: specifies the full path of a directory which should be used for the cache, e.g. `/opt/gravwell/cache/file_follow`. Setting this option to a non-empty string enables ingester caching. If the specified directory does not exist, a new one will be created. Several files and subdirectories will be created in this directory.
- `Max-Ingest-Cache`: Specifies a maximum size (in megabytes) for the cache. Once the cache has reached this size, later entries may be dropped.
- `Cache-Depth`: The cache depth is the maximum number of in-memory entries to maintain before committing entries to disk. This is most important for absorbing bursts of data, as it can prevent costly writes to disk.
- `Cache-Mode`: There are two cache modes: "always", and "fail". When set to "always", the cache is always available to the ingester, even when there are active indexer connections. When set to "fail", the cache is only engaged when all indexer connections are lost. The "always" mode additionally allows an ingester to begin accepting incoming data before any initial indexer connections are made.

The ingest cache is suitable for use with all ingesters except the File Follower. Gravwell recommends against using a cache with the File Follower ingester because the on-disk source files essentially constitute their own cache. The Federator is especially useful when caching is enabled; it can provide an extra level of resilience between the ingesters and the indexers, allowing ingesters to push their entries to the Federator even when the indexer is unavailable.

7.10.1 Hands-on Lab: Ingester Cache

In this lab we will stand up a Simple Relay ingester with cache enabled and send it traffic while the Gravwell indexer is unavailable. First, launch a Gravwell webserver+indexer container:

```
docker run --rm --net gravnet -p 8080:80 -d --name gravwell gravwell:base
```

Log into the web GUI (http://localhost:8080) and leave the page open.

Then make the ingester container:

```
docker run --rm --net gravnet --name ingesters -d \
-e GRAVWELL_CLEARTEXT_TARGETS=gravwell \
gravwell:ingesters /opt/gravwell/bin/gravwell_simple_relay
```

The ingesters container ships with a default Simple Relay configuration which listens for line-delimited entries on port 7777, shown below:

```
[Global]
Log-Level=INFO

[Listener "default"]
    Bind-String="0.0.0.0:7777"
    Ignore-Timestamps=true
    Tag-Name=default
```

Copy the file out:

```
docker cp ingesters:/opt/gravwell/etc/simple_relay.conf /tmp/simple_relay.conf
```

Edit `/tmp/simple_relay.conf` and add the following line to the `[Global]` section:

```
Ingest-Cache-Path=/opt/gravwell/cache/simple_relay
```

Then copy the config file back in and restart the container:

```
docker cp /tmp/simple_relay.conf ingesters:/opt/gravwell/etc/simple_relay.conf
docker restart ingesters
```

Now that the ingester is configured and running, we will pause the indexer container. Run this command on the host:

```
docker pause gravwell
```

Now we open a shell on the ingester container:

```
docker exec -it ingesters /bin/sh
```

From the ingester container's shell prompt, we now use a loop to send 200,000 entries to the ingester:

```
for i in `seq 1 200000`
do
        echo this is an entry $i
done | nc -w 1 localhost 7777
```

The cache directory contains several files and subdirectories. We can see that with the indexer paused, the amount of data in the cache has increased (your output may look different depending on the state the cache was in when you ingested the data):

```
# ls -lRh /opt/gravwell/cache
.:
total 4K
drwxr-x---    4 root     root        4.0K Jul 22 18:49 simple_relay

./simple_relay:
total 12K
drwxr-x---    2 root     root        4.0K Jul 22 18:45 b
drwxr-x---    2 root     root        4.0K Jul 22 18:45 e
-rw-rw----    1 root     root          37 Jul 22 18:49 tagcache

./simple_relay/b:
total 0
-rw-r-----    1 root     root           0 Jul 22 18:49 cache_a
-rw-r-----    1 root     root           0 Jul 22 18:45 cache_b
-rw-------    1 root     root           0 Jul 22 18:45 lock
```

```
./simple_relay/e:
total 2M
-rw-r-----    1 root     root        2.1M Jul 22 18:51 cache_a
-rw-r-----    1 root     root       30.4K Jul 22 18:50 cache_b
-rw-------    1 root     root           0 Jul 22 18:45 lock
```

To flush the cache, we simply unpause the container on the host side:

```
docker unpause gravwell
```

And verify that the cache has been emptied within the ingester container:

```
# ls -lRh /opt/gravwell/cache
.:
total 4K
drwxr-x---    4 root     root        4.0K Jul 22 18:49 simple_relay

./simple_relay:
total 12K
drwxr-x---    2 root     root        4.0K Jul 22 18:45 b
drwxr-x---    2 root     root        4.0K Jul 22 18:45 e
-rw-rw----    1 root     root          37 Jul 22 18:49 tagcache

./simple_relay/b:
total 0
-rw-r-----    1 root     root           0 Jul 22 18:49 cache_a
-rw-r-----    1 root     root           0 Jul 22 18:45 cache_b
-rw-------    1 root     root           0 Jul 22 18:45 lock

./simple_relay/e:
total 0
-rw-r-----    1 root     root           0 Jul 22 18:53 cache_a
-rw-r-----    1 root     root           0 Jul 22 18:53 cache_b
-rw-------    1 root     root           0 Jul 22 18:45 lock
```

Execute a search on the default tag (`tag=default`) to verify that the entries were properly ingested, as in Figure 7.9

To clean up after the experiment, simply run:

```
docker kill $(docker ps -a -q)
```

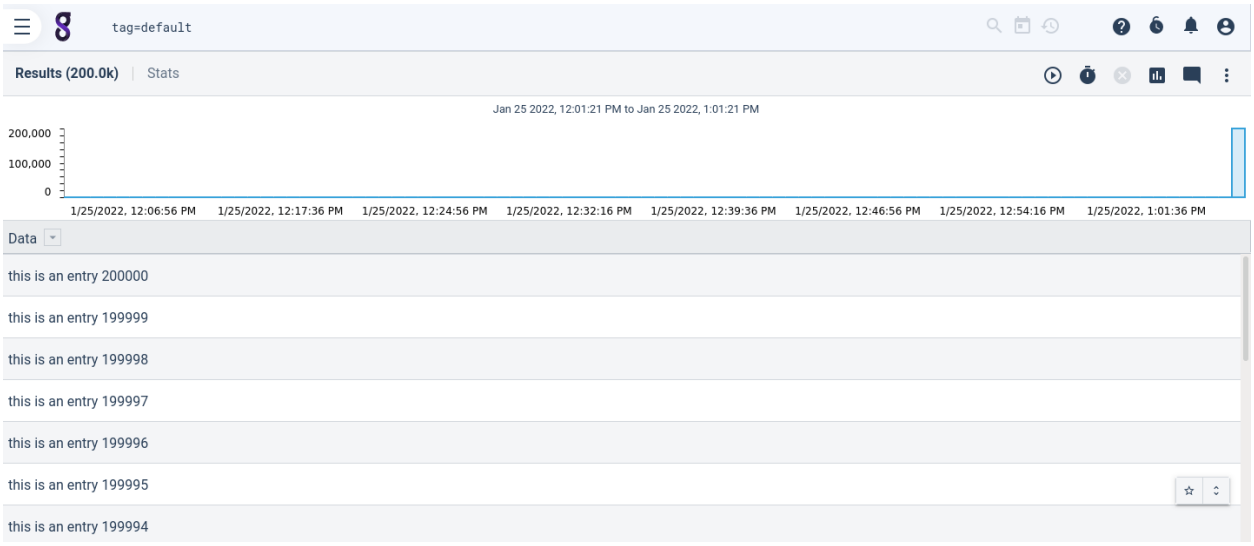


Figure 7.9: Previously-Cached Entries

7.11 Ingest API and Source Code

Gravwell's ingest library is open-sourced for public use at github.com/gravwell/gravwell/ingest. This makes it relatively easy to write a custom ingester for a specific use case completely independent of Gravwell Inc. The open-source ingesters found at github.com/gravwell/gravwell/ingesters provide useful examples of real-world ingesters, but for the sake of demonstration this document will describe the basic steps required to ingest data, including code samples.

7.11.1 Configuring and Starting the Ingest Muxer

The Ingest Muxer is the standard way to connect to one or more Gravwell indexers. If multiple indexers are specified, it will multiplex incoming entries across all indexers evenly, hence the name. To instantiate a muxer, define a UniformMuxerConfig:

```
targets := []string{ "tcp://10.0.0.1:4023", "tcp://10.0.0.2:4023"}
secret := "IngestSecrets"
tags := []string{ "default" }
ingestConfig := ingest.UniformMuxerConfig{
    Destinations: targets,
    Tags:         tags,
    Auth:         secret,
}
```

There are three essential arguments in the muxer config:

- `Destinations`: a list of strings describing how to connect to an indexer, e.g. "tcp://10.0.0.1:4023".
- `Tags`: a list of strings containing the tags the ingester will use
- `Auth`: a string containing the shared secret used to authenticate with the indexers.

With the UniformMuxerConfig defined, instantiate and start the muxer:

```
    igst, err := ingest.NewUniformMuxer(ingestConfig)
    if err != nil {
        log.Fatalf("Failed build our ingest system: %v\n", err)
    }
    defer igst.Close()
    if err := igst.Start(); err != nil {
        log.Fatalf("Failed start our ingest system: %v\n", err)
    }
```

After calling `Start()` on the muxer, one should typically call `WaitForHot` to wait until at least one indexer connection is live. The argument to `WaitForHot` is a timeout in case no indexer connections are established:

```
    if err := igst.WaitForHot(60 * time.Second); err != nil {
        log.Fatalf("Timedout waiting for backend connections: %v\n", err)
    }
```

7.11.2 Creating and Uploading Entries

Once the muxer has been started and `WaitForHot` has returned successfully, entries can be sent to the indexer(s). Note, however, that indexers use a mapping of string tag names to numeric tag IDs, and that entries sent to the indexer must use the `numeric` tag IDs, not string tags. Thus, we first query the ingest muxer for the tag ID of the "default" tag:

```
    defaultTagID, err := igst.GetTag("default")
    if err != nil {
```

```
        log.Fatalf("Failed to get tag: %v", err)
    }
```

Having acquired the tag, we are now able to build an Entry:

```
    ent := entry.Entry{
        TS:   entry.Now(),
        SRC:  net.ParseIP("127.0.0.1"),
        Tag:  defaultTagID,
        Data: []byte("This is my test data!"),
    }
```

The components of an Entry are:

- `TS` The timestamp attached to the entry. Although this example uses the current time, it can be set to anything. The timestamp should generally indicate when the *data* was generated, not when the *entry* was created.
- `SRC`: An IP address to represent the source of the data. We are using 127.0.0.1 for this example, but any IPv4 or IPv6 address is valid.
- `Tag`: A numeric tag ID as derived from the `GetTag` function.
- `Data`: A slice of bytes that makes up the actual *content* of the entry. This can be literally anything.

With the Entry built, the only thing remaining is to send it to the indexer:

```
    if err := igst.WriteEntry(&ent); err != nil {
        log.Fatalf("Failed to write entry: %v", err)
    }
```

7.11.3 Cleaning Up/Shutting Down

An ingester will typically start up, build its muxer, and then go into a loop building and uploading Entries until it either runs out of data or receives a signal from the operating system that it should shut down. In order to shut down nicely, call the `Sync` and `Close` functions on the muxer:

```
if err := igst.Sync(time.Second); err != nil {
    log.Fatalf("Failed to sync: %v\n", err)
}
igst.Close()
```

7.12 Permissions and Port Binding

Gravwell is designed to execute with as few privileges as possible. The shell and Debian installers will create an unprivileged user and group named gravwell and create a directory structure in `/opt/gravwell` which is owned by the gravwell user. Most of the services don't need any special privileges, however if we want to be able to serve the Gravwell GUI on traditional HTTP and HTTPS ports we will need the ability to bind to port 80 and port 443, which are privileged system ports. Modern Linux kernels have the ability to assign capabilities[9] to executables. Capabilities grant specific privileges that would normally be denied to the executing user. Gravwell uses these capabilities to enable the webserver to bind to privileged ports without requiring that we execute as an elevated user or group.

When debugging broken Gravwell installations there are a few things to always check:

1. The ownership of files in `/opt/gravwell` and any well locations. They should be owned by `gravwell:gravwell`

[9] http://man7.org/linux/man-pages/man7/capabilities.7.html

2. The capabilities assigned to binary files.
   (a) The webserver needs CAP_NET_BIND_SERVICE
   (b) The network capture ingester needs CAP_NET_RAW
3. Check logs for critical error messages.
   (a) `/opt/gravwell/log`
   (b) `/opt/gravwell/log/web`
   (c) `/opt/gravwell/log/crash`
   (d) `/dev/shm/`

The most common breakages are when users move shards or attempt to manually start a Gravwell component as `root`; this causes Gravwell to assign ownership of files and directories to the root user and group. When Gravwell is then started correctly by systemd, the Gravwell processes (running as the `gravwell` user) won't have access to the files and folders they need.

7.12.1 Hands-on Lab: Permissions and Port Binding

Docker typically just executes everything as root, so we will be using a new container that actually uses a proper user and group to execute Gravwell components. Start by cleaning up the environment:

```
docker kill $(docker ps -q)
```

Ensure the `gravwell:brokenperms` container is loaded (if you don't have the `gravwell:brokenperms` image, see Section 2.5 for instructions on how to load it), then start it:

```
docker run -d --net gravnet -p 8080:80 --rm \
--name test gravwell:brokenperms
```

Check the GUI (http://localhost:8080), are we able to access Gravwell? Is the container up?

Let's grab a shell within the container as the root user and start poking around:

```
docker exec -it --user root test /bin/bash
```

The goal is to fix the installation and get the Gravwell components to start correctly. Start by answering a few questions:

1. Which services are not starting?
2. Where are the pertinent log files?
3. What other locations contain Gravwell logs?
4. What are the permissions inside `/opt/gravwell/`?
   (a) What should they be?
5. What are the capabilities assigned to each Gravwell service binary?
   (a) What should they be?

To clean up after the experiment, simply run:

```
docker kill $(docker ps -a -q)
```

7.13 Gravwell and Systemd

Gravwell installers assume the availability of Systemd. While there are other init services out there, Systemd has steadily become the init system of choice for most popular Linux distributions. Systemd controls how processes start, how failures are handled, and how to shutdown processes using unit files[10][11]. Unit files can be installed in several locations, but Gravwell places its unit files in `/etc/systemd/system/`. The unit files are copied to the target folder, then registered and started using `systemctl enable <service name>` and `systemctl start <service name>`.

This is the unit file for the Gravwell indexer that is shipped for version 3.3.5:

```
[Install]
WantedBy=multi-user.target

[Unit]
Description=Gravwell Indexer Service
After=network-online.target
OnFailure=gravwell_crash_report@%n.service

[Service]
Type=simple
ExecStart=/opt/gravwell/bin/gravwell_indexer -stderr %n
ExecStopPost=/opt/gravwell/bin/gravwell_crash_report -exit-check %n
WorkingDirectory=/opt/gravwell
Restart=always
User=gravwell
Group=gravwell
StandardOutput=null
StandardError=journal
LimitNPROC=infinity
LimitNOFILE=infinity
TimeoutStopSec=120
KillMode=process
KillSignal=SIGINT
```

By default we do not limit the number of files or processes that Gravwell has access to. Large systems can spread out and run many threads and while managing potentially thousands of shards. However, if your Gravwell is co-resident with other important services it may be desirable to limit access to resources. We do not recommend limiting the number of open files or processes, but limiting CPU access and memory usage may be desirable. CPU and memory can be limited using the `CPUQuota` and `MemoryMax` parameters in the `[Service]` block. IO and network resources can also be limited using `IO`[12].

7.14 Gravwell and Docker

Gravwell supports Docker as a first class citizen (as you can tell by our heavy use for this training). Most components can use environment variables and Docker secrets for configuration and process control. Our published Docker container uses a process control daemon called manager[13] which can monitor environment variables that tell it to disable services. The available control parameters are fully documented in Gravwell's web documentation[14], but let's dive into controlling Gravwell processes and control parameters using docker.

[10] https://access.redhat.com/documentation/en-us/red_hat_enterprise_linux/7/html/system_administrators_guide/sect-managing_services_with_systemd-unit_files

[11] https://www.digitalocean.com/community/tutorials/understanding-systemd-units-and-unit-files

[12] https://www.freedesktop.org/software/systemd/man/systemd.resource-control.html

[13] https://github.com/gravwell/manager

[14] https://docs.gravwell.io/#!configuration/docker.md

Gravwell services have a hierarchy of configuration sources when starting. Configuration parameters in a configuration file are checked first; if a mandatory configuration parameter is empty, it will then potentially look for the configuration parameter in an environment variable. The environment variable is a two step lookup. First it checks if an environment variable is set indicating that we should use secrets. Docker secrets are only available when operating in swarm mode[15]. Kubernetes[16] and OpenShift[17] have similar systems that are compatible with Docker secrets and thus Gravwell. Rather than attempt to get a full Docker swarm environment up, configuration parameters can also be set as regular Docker environment variables. The core Gravwell package supports the following environment/secret configuration parameters:

```
GRAVWELL_REMOTE_INDEXERS
GRAVWELL_REPLICATION_PEERS
GRAVWELL_INDEXER_UUID
GRAVWELL_WEBSERVER_UUID
GRAVWELL_INGEST_AUTH
GRAVWELL_CONTROL_AUTH
GRAVWELL_SEARCHAGENT_AUTH
GRAVWELL_DATASTORE
GRAVWELL_DATASTORE_LISTEN_ADDRESS
GRAVWELL_DATASTORE_LISTEN_PORT
```

If `GRAVWELL_ENABLE_DOCKER_SECRETS` is set to TRUE, then Gravwell will attempt to load the configuration parameters from the secrets system, otherwise it looks for the value in an environment variable. We use the environment variable method of configuration extensively throughout this training. Run `docker inspect gravwell:base` and examine the `Config` and `Env` sections. Those environment variables can be overridden when building a container using the `-e` flag. For example, `docker run -d --name test -e GRAVWELL_INDEXER_UUID=611757dc-4b54-11e9-acf8-aba211cc704f gravwell:base` will start the indexer with a specific UUID, preventing it from generating and assigning a random value.

7.14.1 Hands-on Lab: Gravwell and Docker

For this lab we will examine how to use Docker to modify core configuration variables at runtime while deploying a base `gravwell.conf` file. To start, let's clean the working environment:

```
docker kill $(docker ps -q)
```

Now create a new container named test using the gravwell:base image. When you create the container, override the default `Ingest-Auth`, `Control-Auth`, and `Search-Agent-Auth` auth tokens with something unique. Once the container has been created using `docker create`, log into the GUI and check that everything came up, then try ingesting some data using JSON generator; be sure to set the appropriate authentication secrets!

```
docker run --net gravnet --rm -it gravwell:generators \
/jsonGenerator -clear-conns test \
-ingest-secret MY_SECRET
```

Lab Questions

1. How can we check the environment variable overrides using Docker?
2. Why is it insecure to use environment variables to configure authentication tokens?
3. Why is this configuration system essential for deployment using something like Kubernetes or OpenShift?

To clean up after the experiment, simply run:

```
docker kill $(docker ps -a -q)
```

[15] https://docs.docker.com/engine/swarm/secrets/
[16] https://kubernetes.io/docs/concepts/configuration/secret/
[17] https://docs.openshift.com/container-platform/3.5/dev_guide/secrets.html

Chapter 8

Automation

Gravwell provides several utilities to enable automated operations. At the most basic level, users can schedule searches to be executed at specific times, for example every morning at 1:00. They can also schedule *flows* and *scripts*, which can execute multiple searches, sift and parse search results, and send notifications via email or HTTP. Flows are designed using a drag-and-drop graphical interface and are generally preferred over scripts, which should be reserved for legacy situations or very particular usecases which flows cannot cover.

These scheduled operations are run by the **search agent**, a separate program which connects to the Gravwell webserver as a client. This chapter describes the search agent, scheduled searches, flows, and scripts.

8.1 Configuring User Email Settings

In order to send emails from scheduled scripts, each user must input settings for their preferred email server. This will allow Gravwell to act as an SMTP client and send emails on the user's behalf. The email configuration page is a sub-tab in the user's account settings (see Figure 8.1).

The Username and Password fields should be filled out with the user's authentication tokens for the *email server*. If TLS is required by the server, set the 'Use TLS' toggle; note that it may be necessary to change the Port to 465 after enabling TLS.

After populating the configuration, hit 'Update Settings' to save the options. 'Test Configuration' should then become clickable. Click it to bring up the email testing dialog, as seen in Figure 8.2.

Change the "From" and "To" addresses to the user's own email address, then click 'Send Test Email'. Gravwell will attempt to use the mail server to send an email to the user. If the email arrives, the email server has been correctly configured. It may be necessary to check spam folders.

If the email does not arrive, check the Gravwell webserver logs, located in `/opt/gravwell/log/web` by default.

8.2 The Search Agent

The search agent is the Gravwell component which actually handles the execution of scheduled searches and scripts. It is a separate process which connects to a Gravwell webserver as a client, obtains a list of scheduled searches/scripts, and runs them when scheduled.

The search agent is shipped in the main Gravwell installer and should be installed by default in most situations. If for some reason the search agent is not enabled, a notification will appear in the Gravwell web interface warning of that fact (Figure 8.3).

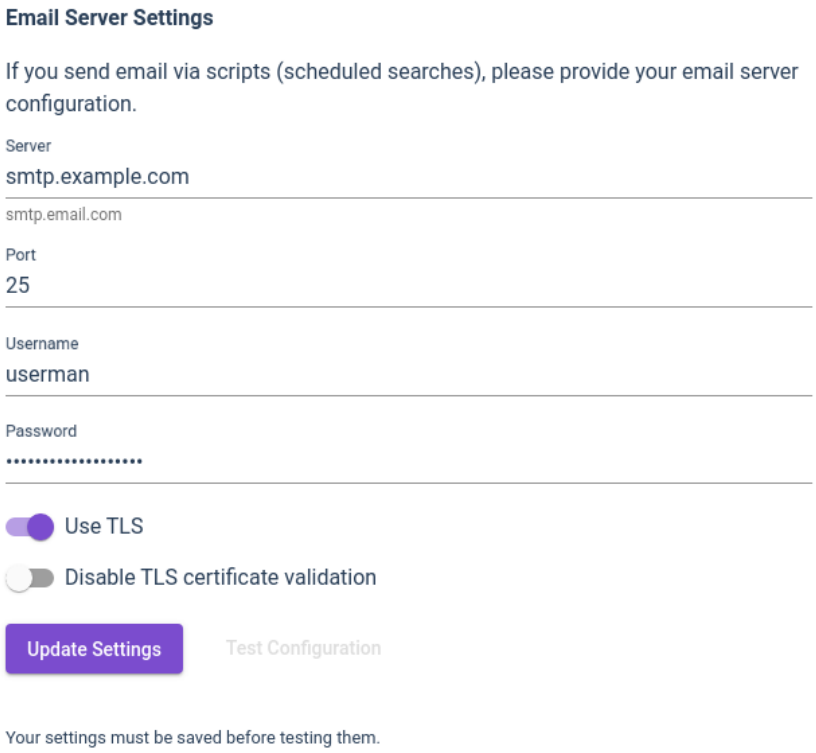


Figure 8.1: Email preferences dialog

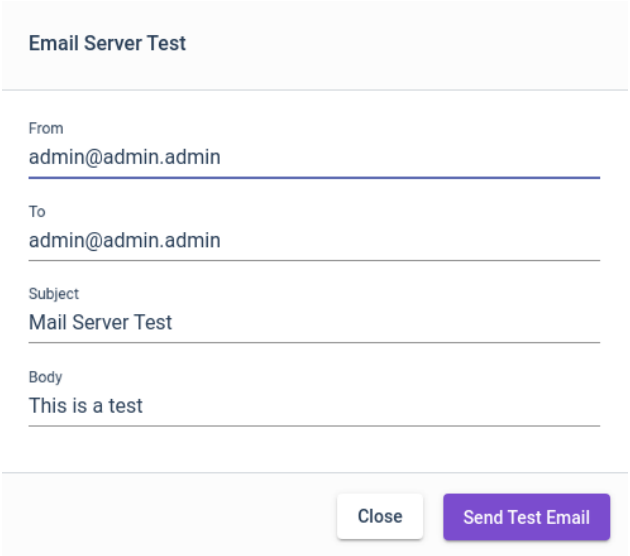


Figure 8.2: Email testing dialog

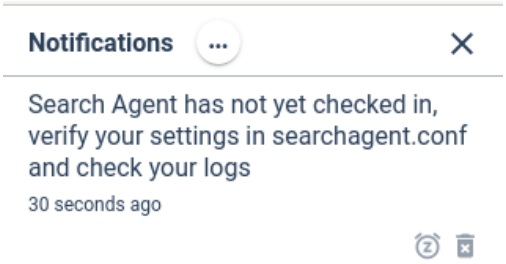


Figure 8.3: Search agent warning

Note that when using multiple Gravwell webservers, the search agent should be disabled on all but one of them to avoid conflicts and superfluous search executions.

8.2.1 Disabling the Search Agent

To disable the search agent on a server that uses Systemd:

```
systemctl stop gravwell_searchagent.service
systemctl disable gravwell_searchagent.service
```

The Docker images provided by Gravwell do not use Systemd; instead, they use a Gravwell-implemented process manager called "manager"[1]. To disable the search agent on such a container, pass the argument `-e DISABLE_SEARCHAGENT=TRUE` when creating the container, e.g.:

```
docker run --rm --net gravnet -p 8080:80 -d \
-e DISABLE_SEARCHAGENT=TRUE --name gravwell gravwell:base
```

8.2.2 Search Agent Configuration

Although the search agent ships with the indexer and webserver components, it uses its own configuration file found at `/opt/gravwell/etc/searchagent.conf`. While the defaults set at install time should be sufficient, this section describes the configuration options in case changes are needed.

Webserver-Address

The `Webserver-Address` parameter should be an IP or hostname and a port which point to the Gravwell webserver. If no port is specified, port 443 is used by default (80 if HTTPS is disabled). The following are all valid Webserver-Address values:

```
Webserver-Address=10.0.0.1:80
Webserver-Address=127.0.0.1:443
Webserver-Address=192.168.0.1
Webserver-Address=gravwell-webserver.example.com
```

If Webserver-Address is specified multiple times, the search agent will multiplex scheduled searches across all listed servers, to distribute load.

Search-Agent-Auth

The `Search-Agent-Auth` parameter specifies the token which is used to authenticate with the webserver. This can be any string, but it *must* correspond with the `Search-Agent-Auth` parameter defined in the webserver's `gravwell.conf`!

[1] https://github.com/gravwell/manager

Insecure-Use-HTTP

Setting the `Insecure-Use-HTTP` parameter to "true" instructs the search agent to connect to the Gravwell webserver using unencrypted HTTP. This should only be used on trusted internal networks or on the local loopback interface!

Insecure-Skip-TLS-Verify

Setting the `Insecure-Skip-TLS-Verify` parameter to true instructs the search agent to ignore invalid TLS certificates when connecting to the webserver.

Log-File

`Log-File` specifies a file where the search agent should write its logs. If not set, the search agent will write logs to standard error.

Log-Level

`Log-Level` sets the severity level at which log entries will actually be written to the log, thus setting `Log-Level=WARN` ensures only warnings, errors, and critical messages will be sent to the log. The available levels are OFF, DEBUG, INFO, WARN, ERROR, and CRITICAL.

Max-Script-Run-Time

The `Max-Script-Run-Time` parameter specifies how long, in minutes, a given scheduled script is allowed to execute. If set to 0, scripts can run for an unlimited length of time. There are two things to consider when setting this:

1. A script which runs indefinitely only interferes with itself; other scheduled searches and scripts will run regardless.
2. Many buggy scripts will automatically time out, because a script must also execute at least one of the Gravwell-defined scripting functions every 30 seconds. A hung script will quickly be terminated.

8.2.3 Scheduling Searches

Users can schedule searches to run at regular intervals. This enables several useful possibilities, such as automatically updating lookup tables (e.g. MAC address to IP mappings) or executing a very detailed / long-running search every morning at 6 a.m. to have the results ready when employees arrive.

Creating a Scheduled Search

To create a scheduled search, select the 'Scheduled Searches' page from the menu in the GUI, then click the 'Add' button in the upper right corner of the page. A form will pop up, as shown in Figure 8.4.

Enter the desired query in the box labeled 'Type in a query', then use the timeframe picker to choose how far back the search should run. Assign the scheduled search a name and a description.

The 'Scheduling' field requires some explanation. This field defines a cron-formatted schedule to specify when the search should run. Clicking the 'Cron Helper' link will open a useful website to experiment with cron schedules. The following are all valid schedules:

- Run every minute: `* * * * *`
- Run every 10 minutes: `*/10 * * * *`
- Run every hour: `0 * * * *`
- Run every day at 2 a.m.: `0 2 * * *`
- Run once a week at 7 P.M.: `0 19 */7 * *`

When all the fields have been populated, click 'Save' to save the scheduled search. The search agent will soon discover this new search and will execute it on schedule.

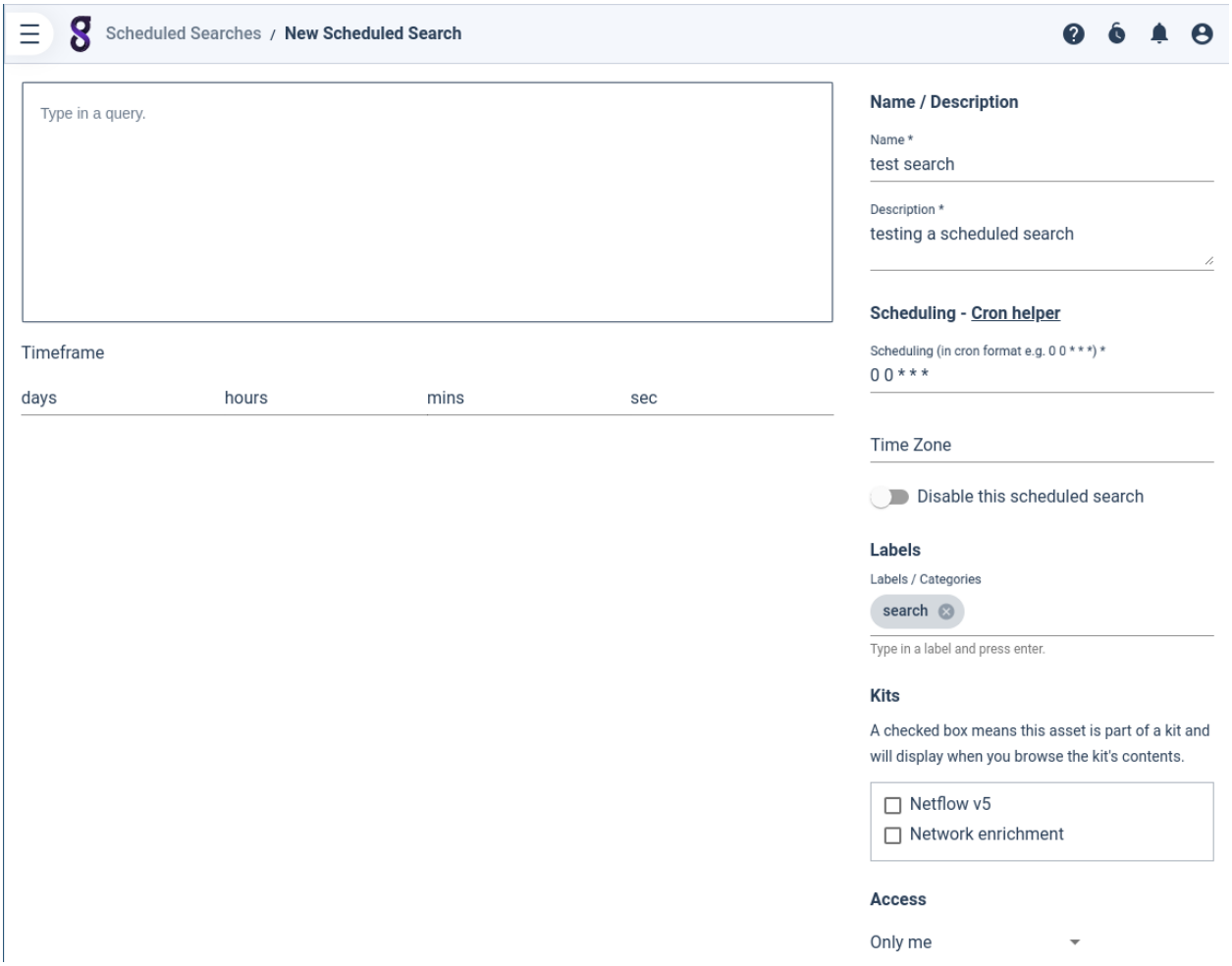


Figure 8.4: New scheduled search form

Viewing Scheduled Search Results

Once a scheduled search has been executed at least once, the results of the most recent search execution are available for review. Although the search results appear in the 'Persistent Searches' page, the simplest way to view the results for a particular scheduled search is to select the 'Run last search' icon on the tile for that scheduled search within the 'Scheduled Searches' page (Figure 8.5).

This will open the most recent results from that scheduled search. Note that when the search agent runs a scheduled search, it deletes the previous run's results when the new search completes. This prevents old searches from cluttering the disk.

8.2.4 Hands-On Lab

This lab will demonstrate how a scheduled search can automatically update lookup tables. First, create a Gravwell webserver+indexer container:

```
docker run -d --rm --net gravnet -p 8080:80 --name gravwell gravwell:base
```

Then we'll use the ingesters container (make sure you've loaded the ingesters container image as described in Section 2.5) to import some DHCP data:

```
cd ~/gravwell_training/Automation/Lab-Scheduled

docker run -v $PWD/data:/tmp/data --rm -i --net gravnet \
gravwell:ingesters /opt/gravwell/bin/reimport -rebase-timestamp \
-clear-conns gravwell:4023 -i /tmp/data/dhcp.json.gz -import-format json \
-tag-override syslog
```

Log into the web GUI (http://localhost:8080). A simple search on the "syslog" tag over the last two weeks should show some results similar to Figure 8.6.

These DHCP server logs can be used by a scheduled search to build a lookup table for search enrichment, mapping MAC addresses and IP addresses to hostnames. Go to the Scheduled Searches page, click 'Add', and fill in the form with the following:

- **Query:**

  ```
  tag=syslog regex "HOSTNAME: (?P<hostname>\S+) on (?P<ip>\S+) at (?P<mac>\S+)"
  | unique hostname ip mac
  | table -save hosts hostname ip mac
  ```

- **Timeframe:** `14 days`
- **Name:** `hostsfile`
- **Description:** `update hosts file`
- **Scheduling:** `* * * * *`

This scheduled search will run every minute, searching over the last 14 days to extract MAC:IP:hostname mappings. It will use the table renderer's `-save` flag to save the results in a resource named "hosts".

Within a minute, a resource named "hosts" should appear on the Resources page. We will test this lookup table by attempting to look up the hostname which corresponds to the IP requested in DHCPREQUEST messages. Here is a sample DHCPREQUEST message:

```
<190> 11:15:02 apu dhcpd:  DHCPREQUEST for 192.168.2.52 from 08:00:27:ca:b2:e7 via igb2
```

From this, we can create a regular expression which extracts the IP address, then use the new lookup table to match that IP address against a hostname in the hosts resource. Figure 8.7 shows the results.

```
tag=syslog regex "DHCPREQUEST for (?P<ip>\S+)"
| lookup -r hosts ip ip hostname as hostname | table
```

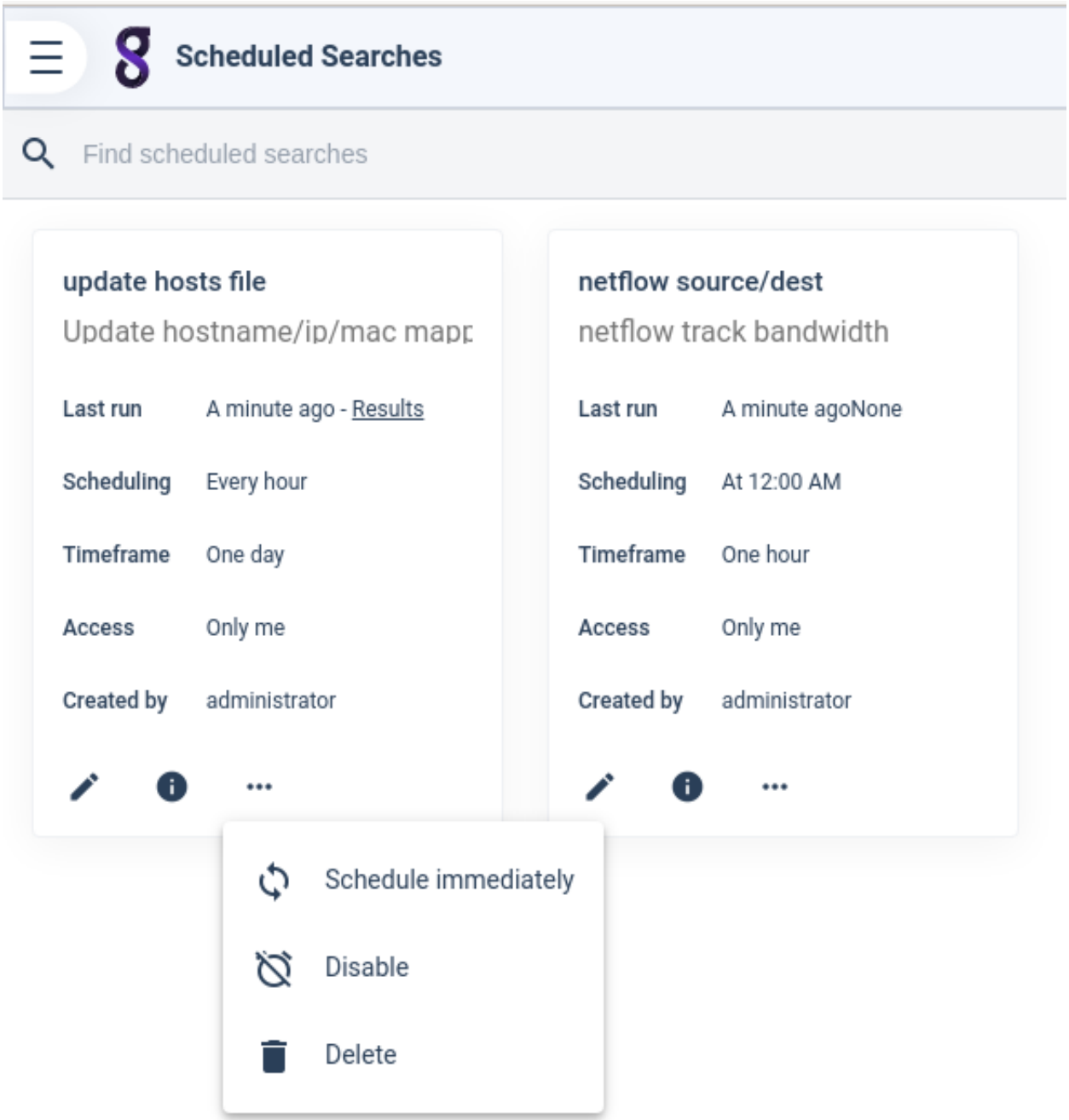


Figure 8.5: The “Run last search” button

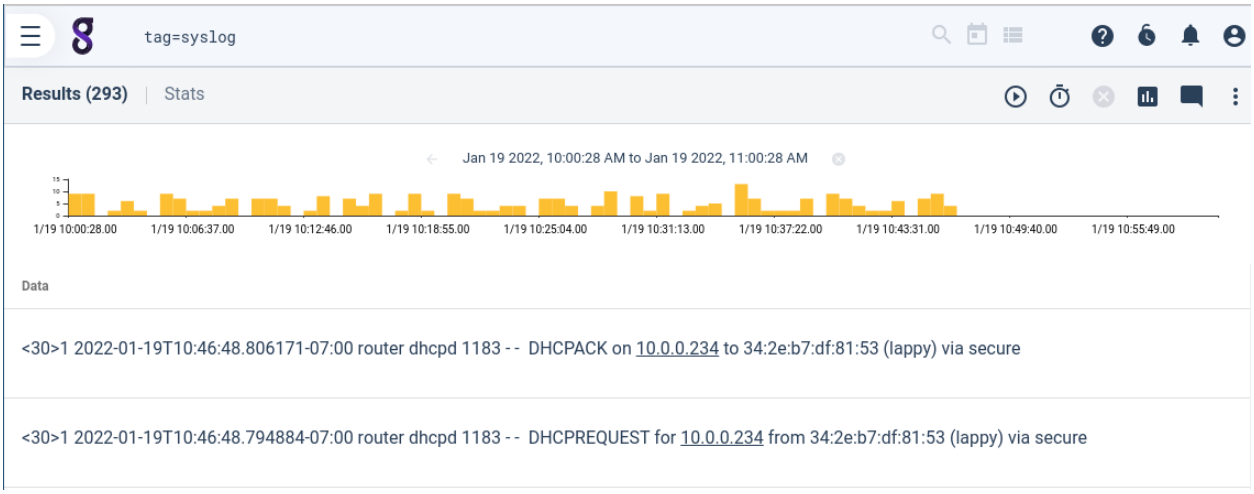


Figure 8.6: Raw DHPC logs

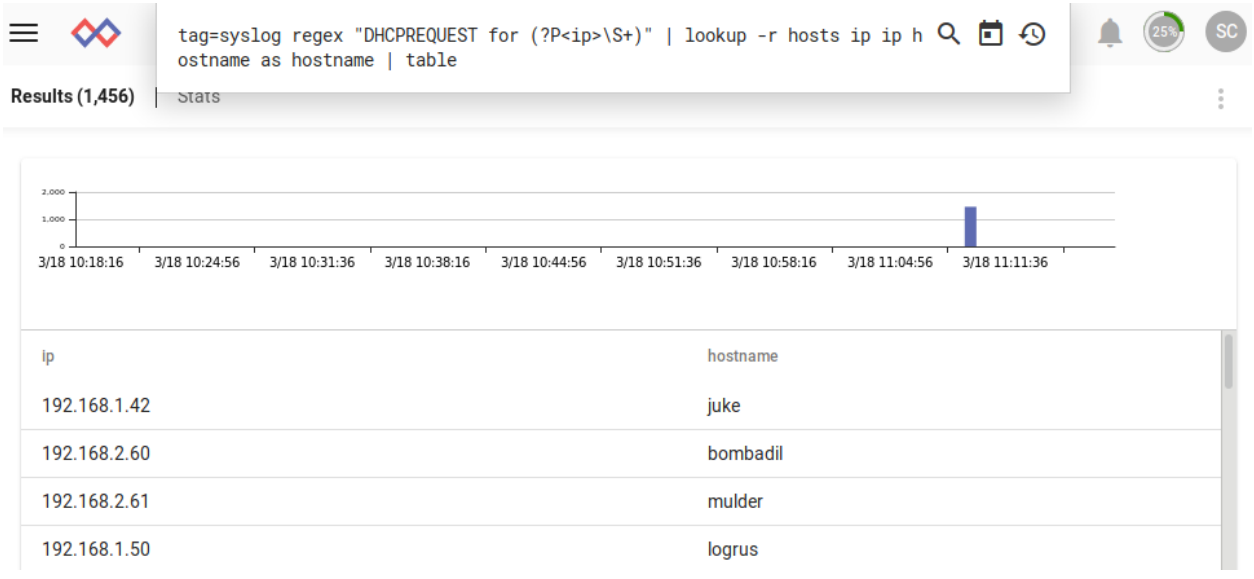


Figure 8.7: Lookup table results

To clean up after the experiment, simply run:

```
docker kill $(docker ps -a -q)
```

8.3 Flows

Flows provide a no-code method for developing advanced automations in Gravwell. A flow consists of one of more *nodes*; each node performs a single action and passes the results (if any) on to the next node(s). By wiring together nodes in a drag-and-drop user interface, users can:

- Run queries
- Generate PDF reports
- Send emails
- Fire off Slack and MS Teams messages
- Re-ingest alerts
- etc.

8.3.1 Flow Concepts

Flows are automations, meaning they are normally executed on a user-specified schedule by the search agent. They can also be run manually through the user interface. The basic process of flow development is:

1. Create a new flow
2. Instantiate nodes in the flow and connect them together
3. Configure nodes
4. Test the flow with debug runs
5. Deploy the flow by setting a schedule & enabling scheduled execution

Nodes

A flow is a collection of *nodes*, linked together to define an order of execution. Each node does a single task, such as running a query or sending an email. Figure 8.8 shows a simple flow of three nodes; the leftmost node runs a Gravwell query, then the middle node formats the results of that query into a PDF document, and finally the rightmost node sends that PDF document as an email attachment.

Figure 8.8: A simple flow with three nodes.

All nodes have a single output socket. Most have only a single input socket, but some nodes which merge *payloads* (see below) have multiple input sockets. One node's output socket may be connected to the *inputs* of multiple other nodes, but each input socket can only take one connection.

Payloads

Payloads are collections of data passed from node to node, representing the state of execution. For instance, the "Run a Query" node will insert an item named "search" into the payload, containing things like the query results and metadata about the search. The PDF node can *read* that "search" item, format it into a nice PDF document, and insert the PDF file back into the payload with a name like "gravwell.pdf". Then the Email node can be configured to attach "gravwell.pdf" to the outgoing email.

Most nodes receive a single incoming payload through a single *input* socket, then pass a single outgoing payload via the *output* socket. In most cases, the outgoing payload will be a modified version of the incoming payload.

The *merge* nodes are exceptions to this general rule. The Stack Merge and Nest Merge nodes take multiple incoming payloads (via multiple input sockets) and merge them into a single output. See Section 8.3.3 for more detailed descriptions of these nodes.

Execution order

Nodes are always executed one at a time. A node can be executed if all nodes upstream of it (its *dependencies*) have executed. If multiple nodes are ready to execute, one will be chosen at random. In Figure 8.9, both the "Run a Query" node and the "HTTP" node are candidates to run first. After the Query node finishes, the If node can execute; when it is done, the Slack Message node may run. We say that the "If" node is *downstream* of the Query node, and the Slack node is *downstream* of both the If and Query nodes.

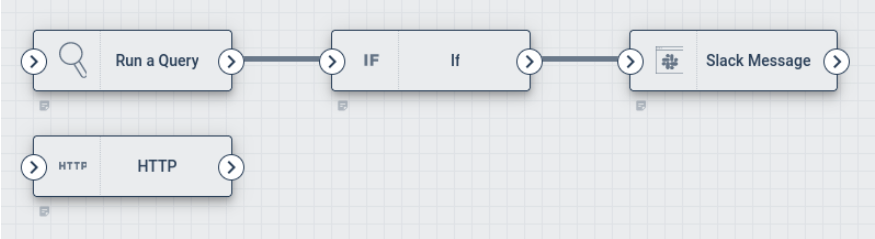


Figure 8.9: An illustration of execution order.

Note that some nodes may block execution of downstream nodes. The `If` node is configured with a boolean logic expression; if that expression evaluates to *false*, none of the If node's downstream nodes are executed. Nodes which can block downstream execution will always have a note to that effect in the online documentation.

8.3.2 The Flow Editor

Flows are created using the flow editor. Although the Gravwell flow editor can be intimidating at first glance, a few minutes' worth of experimentation and exploration should be enough to get started building flows. This section will go through the various components of the UI, explaining each component.

Note: If you're not yet familiar with the basic components of a flow (nodes, sockets, payloads), refer to Section 8.3.1 for an overview.

You can access the flow editor from the Query & Dev Studio interface, found in the Main Menu. Select "Flows" from the left-hand side, as shown in Figure 8.10. From there, you can either start a new blank flow ("Start a New Flow") or instantiate one of the "starter flows" provided by Gravwell.

Selecting either option will take you into the flow editor, the parts of which are marked in Figure 8.11. The *palette* provides a list of available nodes, which can be dragged out into the *canvas*. The *console* provides information about problems with the flow and output from any test runs.

Nodes are instantiated by dragging them from the palette onto the canvas. Once on the canvas, node input and output sockets can be connected, nodes can be re-arranged, etc. Note that the scroll wheel can be used to zoom in and out of the canvas view.

The toolbar contains buttons for quick access to editor functionality. From left to right:

- Flow Designer: shows the flow canvas (default view).
- Info & Scheduling: shows options to set flow name, description, scheduling, sharing, etc.
- Disable scheduling: toggle to quickly enable/disable automatic execution of the flow.
- Save: save the flow.
- Debug: run the flow
- Clear selection: deselects any currently-selected node.

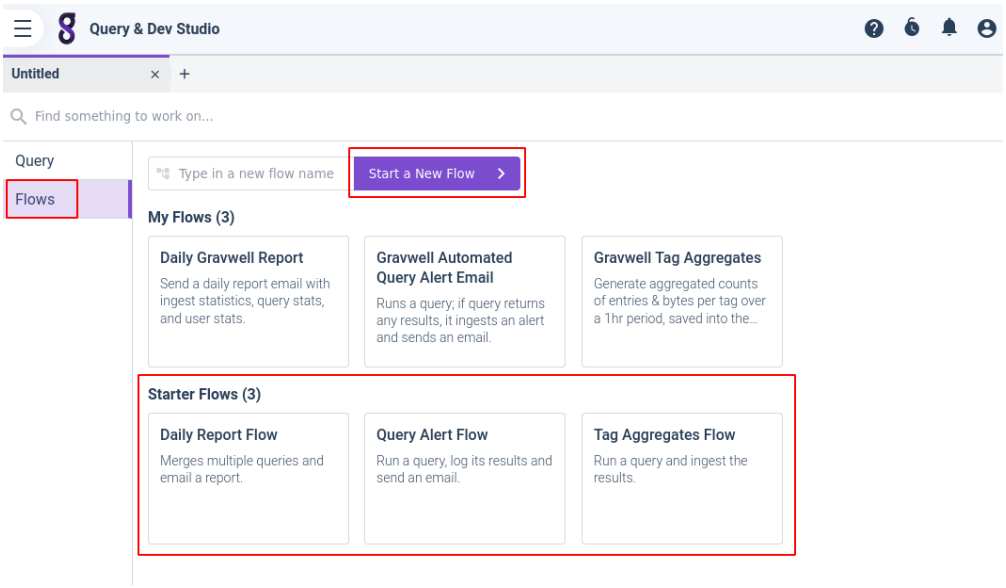


Figure 8.10: Flows in the Query Studio

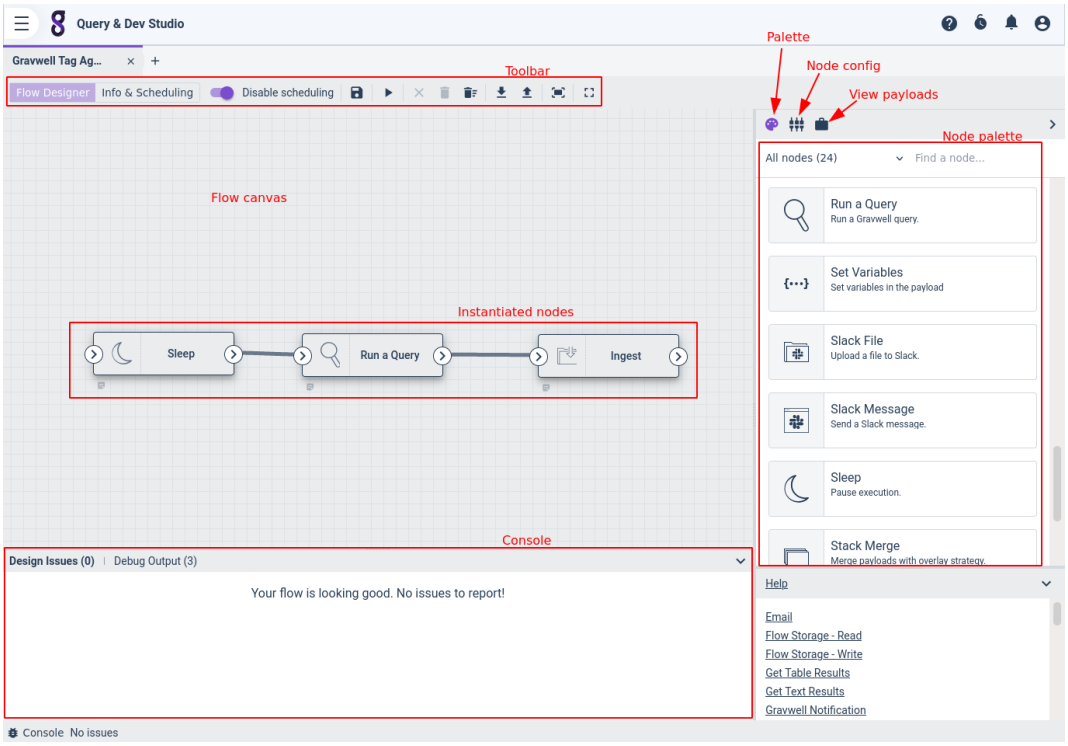


Figure 8.11: Parts of the Flow Editor.

- Delete: delete the selected node.
- Delete all: delete all nodes (requires confirmation).
- Export flow: download the flow specification, for backup or sharing.
- Import flow: upload a previously-exported flow spec.
- Fit all nodes on screen: zoom & center the canvas so that *all* nodes are visible.
- Fullscreen: puts the editor into fullscreen mode.

Configuring Nodes

Once a node has been instantiated by dragging it from the palette to the canvas, it must be configured. Clicking on the node will bring up the configuration pane as seen in Figure 8.12.

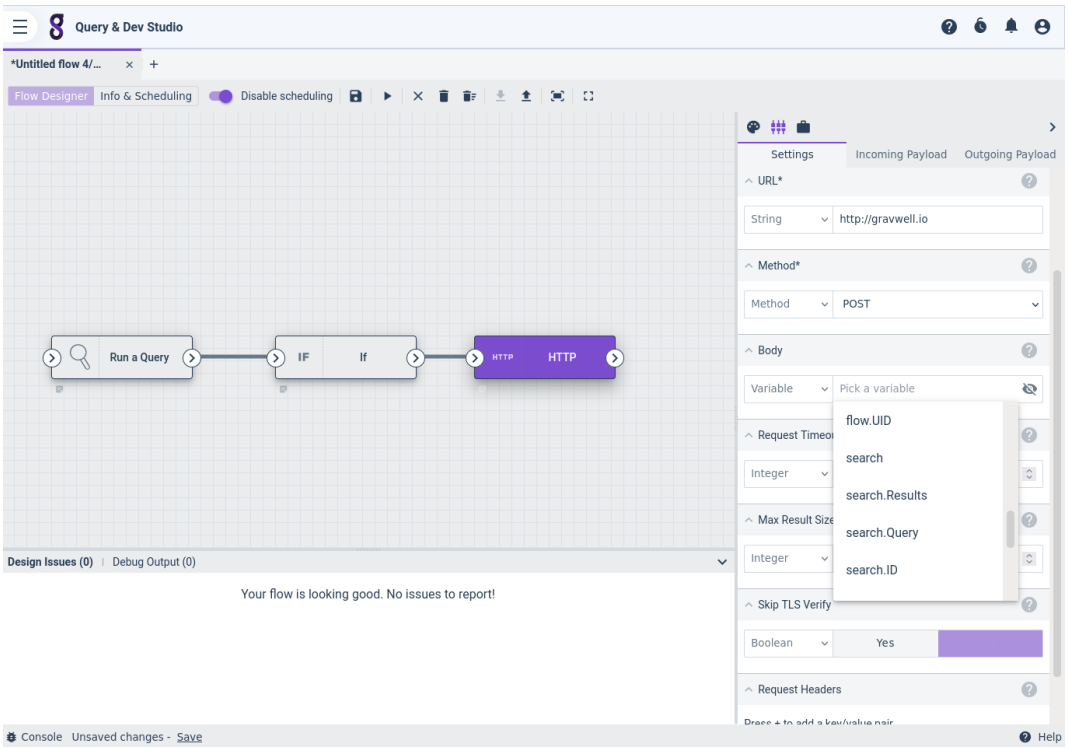


Figure 8.12: Configuring a node.

The HTTP node shown in Figure 8.12 is a particularly complex node with many config options, which serves well for demonstration. Note that the URL and Method fields are marked with an asterisk, indicating that they are required. Note also the drop-down menus for each config option; these allow you to change between entering a constant value (e.g. the string "http://gravwell.io" in the URL config) or selecting a value from the payload as shown with the Body config.

If a node is misconfigured, the console will display a list of problems. In Figure 8.13, we see that the Email node has several config options which are not yet set. As those options are populated, the errors will go away.

Note: You can return to the palette view at any time by clicking the palette icon above the configuration pane.

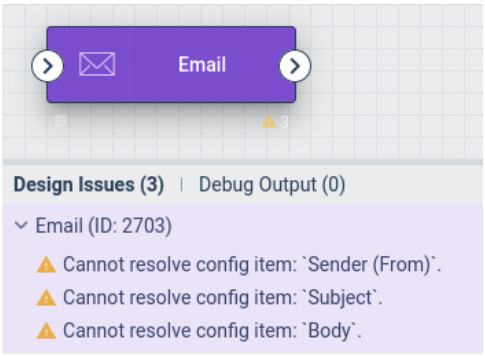


Figure 8.13: Parse errors.

Debugging

Once a flow has been designed and configured, it can be debugged. This will signal the search agent component that it should try executing the flow. To start a debug run, click the "Run flow and debug" button (the "play button") in the toolbar. The user interface will then wait for the search agent to complete its run.

Once the run is complete, the console will have detailed execution information for each node in the "Debug Output" pane. The nodes are listed in order of execution. Clicking on a node in the debug output will bring up a pane showing that node's log output and the actual contents of that node's output payload. In Figure 8.14, we can see that the If node received a payload where `search.Count` was "10", meaning the If node's boolean statement evaluated to true and the HTTP node was allowed to execute:

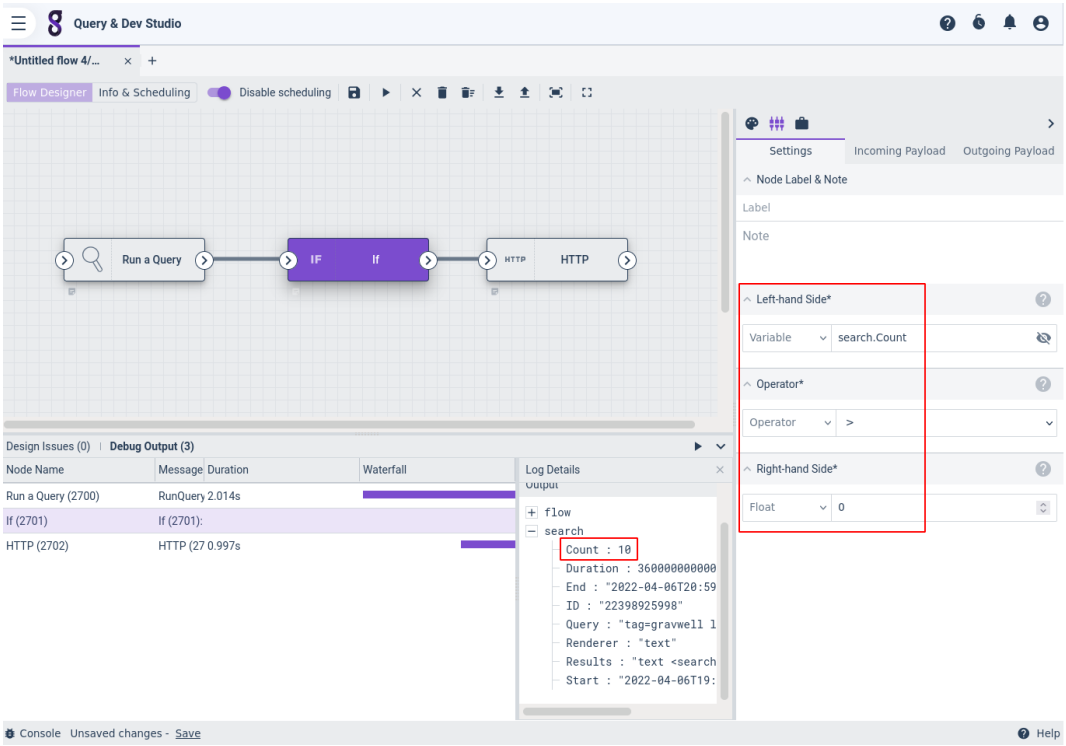


Figure 8.14: Node payload showing If node payload.

If we modify the If node's config so the statement is `search.Count <= 0` and re-run the flow, we'll see that

it now evaluates to false and the HTTP node does not execute (as seen by the empty "Message" column in the Debug Output pane), as shown in Figure 8.15

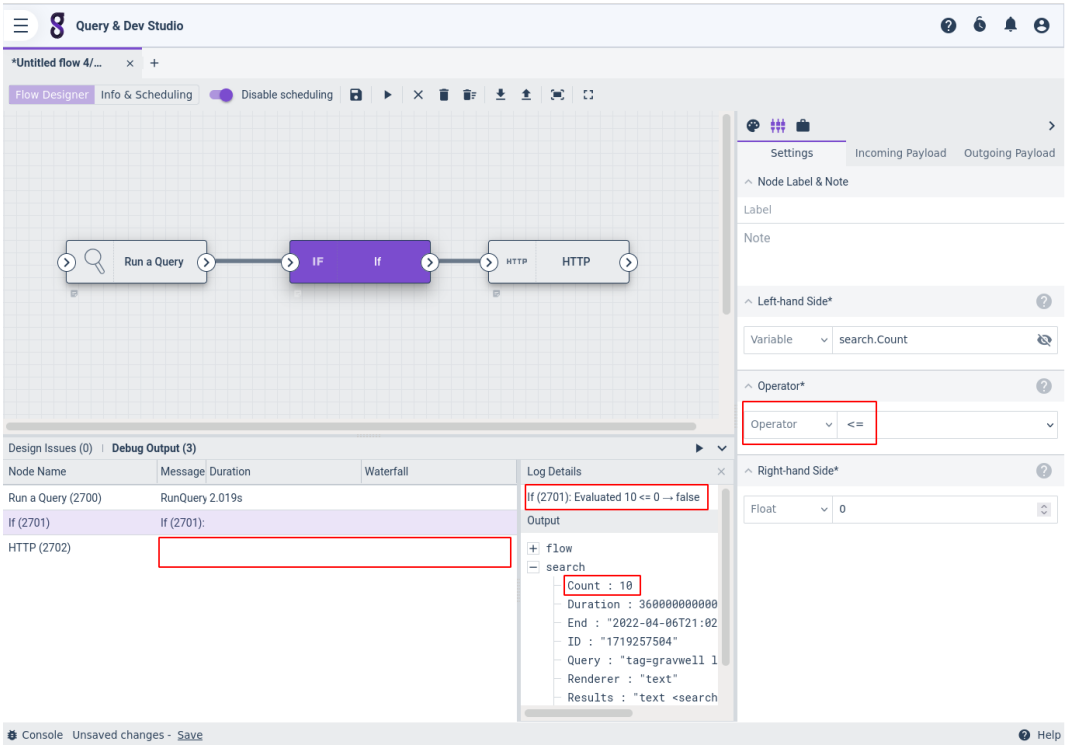


Figure 8.15: Debug run where If node evaluates to false and execution halts.

Info & Scheduling

Once you're happy with a flow, the final step is to give it a schedule and enable it. This is done in the "Info & Scheduling" page, accessible via a button in the toolbar.

You should specify a name and description for the flow, then define a schedule. The schedule is set in cron format[2], which is very flexible but can also be intimidating. There are a few shortcuts for simple cases: `@hourly` runs at the start of every hour, `@daily` at midnight every day, and so on.

Once the schedule is set, toggle the "Disable scheduling" option to enable scheduled executions of the flow. The search agent will then automatically run it on the given schedule.

In-Flow "Sticky" Notes

The "Note" node is a special node used to annotate flows. Unlike other nodes, it plays no role in the execution of the flow; notes exist purely for the convenience of users.

When dragged from the palette, a Note node starts out in a minimized state as seen in Figure 8.17.

When clicked, the note expands and text can be entered, as shown in Figure 8.18.

Clicking the "X" will minimize the note, leaving the start of the text visible as seen in Figure 8.19.

[2]https://cron.help/

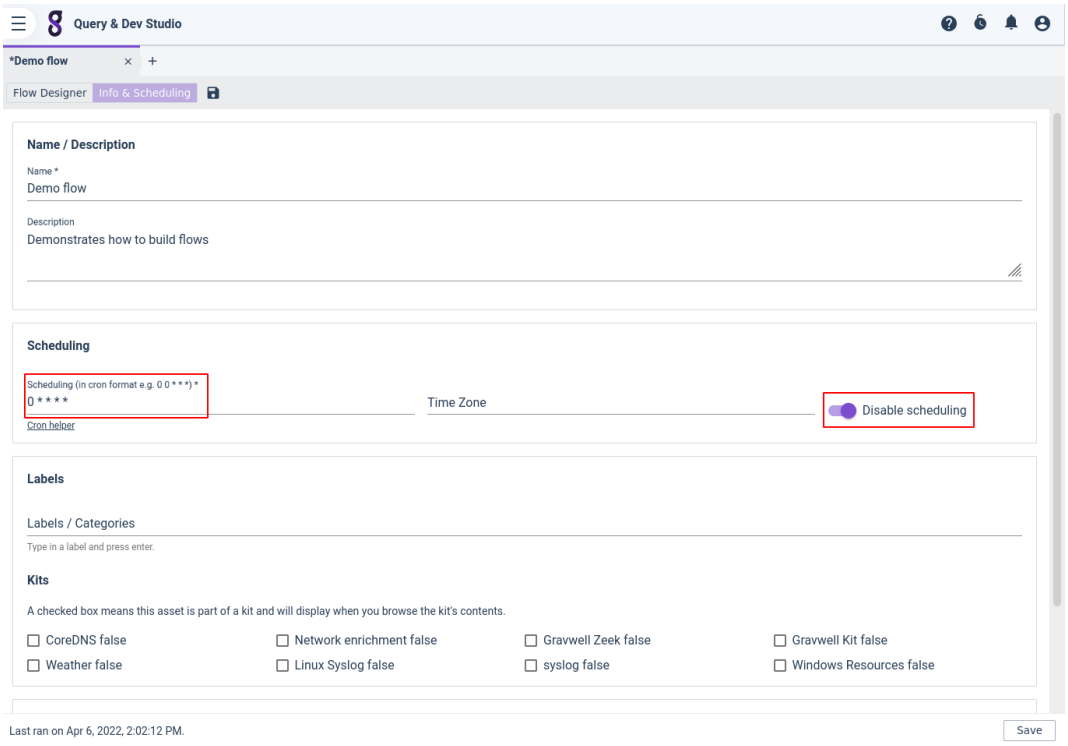


Figure 8.16: Info & scheduling page for flows.

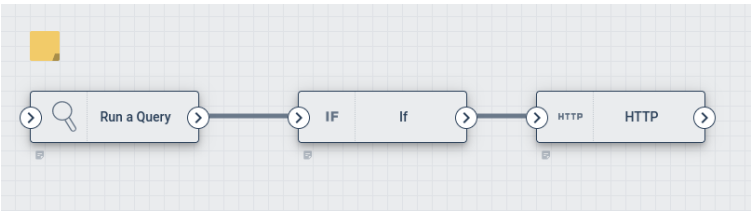


Figure 8.17: A minimized note.

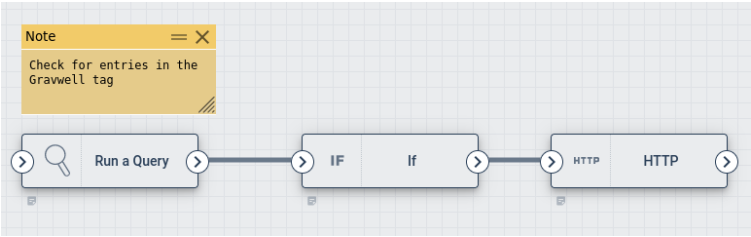


Figure 8.18: An "open" note.

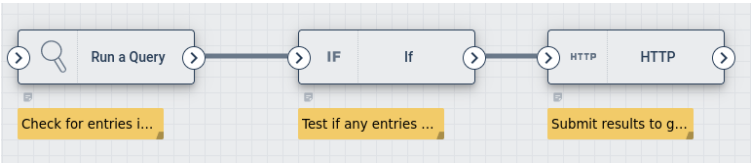


Figure 8.19: Multiple minimized notes with text preview visible.

8.3.3 Nodes

The following nodes are currently implemented:

- Email: send email messages.
- Flow Storage Read: read items from a persistent storage.
- Flow Storage Write: write items into a persistent storage.
- Gravwell Notification: set Gravwell notifications.
- HTTP: do HTTP requests.
- If: perform logical operations.
- Indexer Info: get information about Gravwell indexers.
- Ingest: ingest data into Gravwell.
- Ingester Info: get information about Gravwell ingesters.
- JavaScript: run small snippets of Javascript code for custom operations.
- JSON Encode/Decode: encode and decode JSON.
- Mattermost Message: send a Mattermost message.
- Nest Merge: join multiple input payloads into one.
- PDF: create PDF documents.
- Query Log Ingest: convert search results to alert entries & ingest.
- Read Macros: read Gravwell macros.
- Rename: rename variables in the payload.
- Run a Query: run a Gravwell query.
- Set Variables: inject variables into the payload.
- Slack File: upload a file to a Slack channel.
- Slack Message: send a message to a Slack channel.
- Sleep: pause flow execution for a given period of time.
- Splunk Query: run a Splunk query.
- Stack Merge: join multiple input payloads into one.
- Teams Message: send a Microsoft Teams message.
- Text Template: format text.
- Throttle: limit execution frequency of certain nodes within a flow.

The following nodes tend to be needed only in particular advanced cases:

- Get Table Results: get results from a search using the table renderer.
- Get Text Results: get results from a search using the text renderer.

A selection of nodes are described in greater detail below. Documentation for every individual node is available on the Gravwell documentation website.[3]

[3] https://docs.gravwell.io/#!flows/flows.md

Nest Merge Node

The Nest Merge node can join multiple input payloads into a single output payload. It "nests" each input payload under a different name in the outgoing payload. Figures 8.20 and 8.21 show a simplified example of how this works. Two Set Variable nodes are instantiated, each injecting a variable named "foo"; the first sets the value to `first value` and the other sets the value to `second value`. The outputs of these nodes are fed to the Nest Merge node, which has been configured with two input sockets named `x1` and `x2`. Figure 8.21 shows how the Nest Merge node places the incoming payloads under new top-level names corresponding to the input sockets; thus the `foo: "first value"` output of the first Set Variable node is nested under the name `x1`.

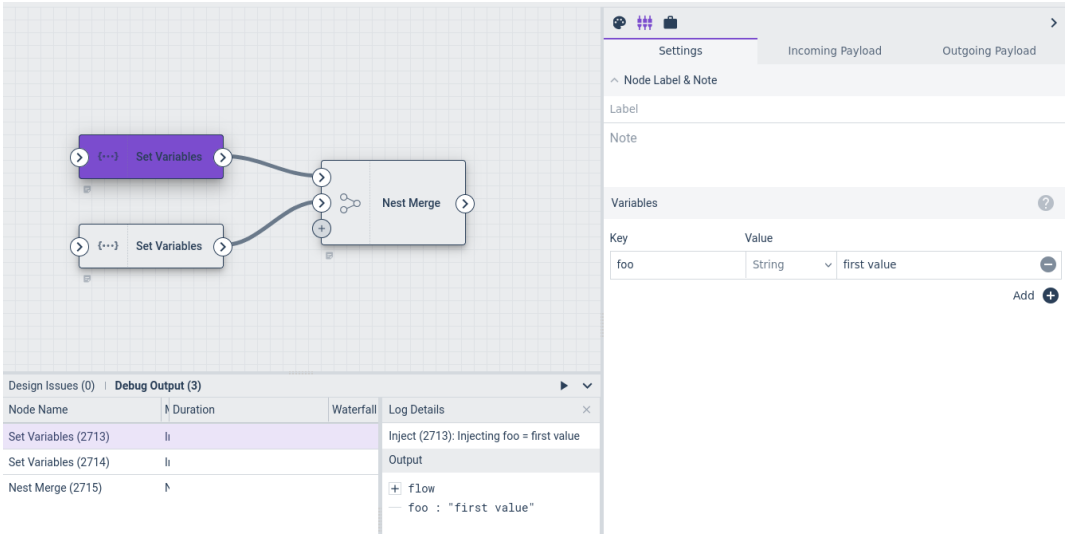


Figure 8.20: Nest merge inputs.

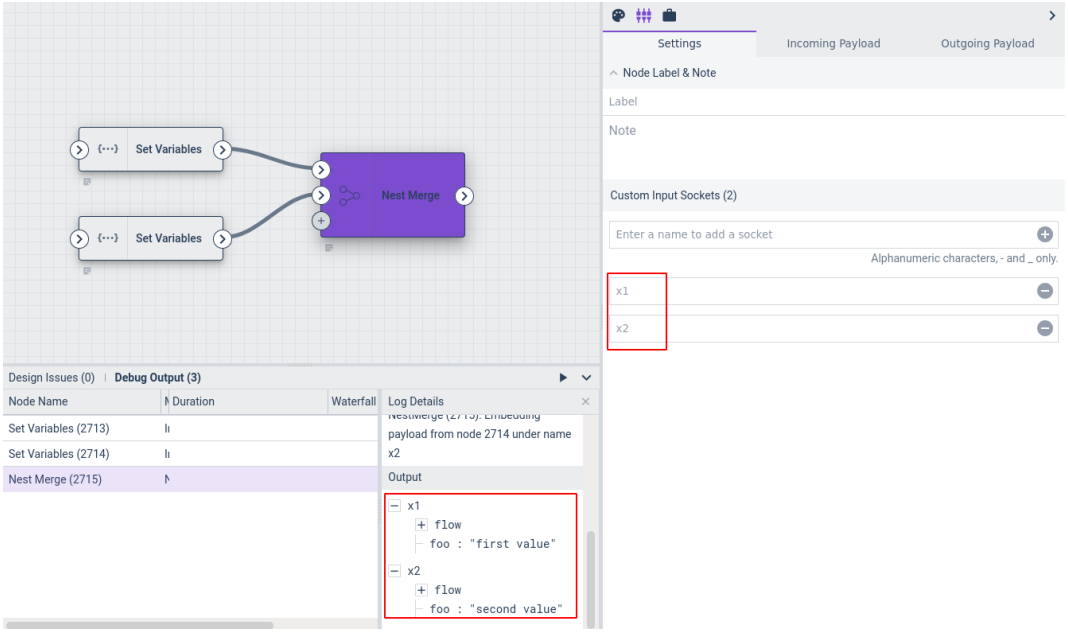


Figure 8.21: Nest merge output.

Stack Merge Node

The Stack Merge node can join multiple input payloads into a single output payload. Where the Nest Merge node "nests" input payloads and thus always preserves the entirety of all inputs, the Stack Merge node "overlays" inputs. That is, it takes the first input payload, then merges in the second payload, *overwriting* any variables with the same name. It repeats this process for all input payloads.

Figures 8.22 and 8.23 show a simplified example of how this works. Two Set Variable nodes are instantiated, each injecting a variable named "foo"; the first sets the value to `first value` and the other sets the value to `second value`. The first node also injects a variable named "x" with a value of `y`.

The outputs of these nodes are fed to the Stack Merge node, which has been configured with two input sockets. Figure 8.23 shows how the Stack Merge node overwrites values; thus the `foo: "first value"` output of the first Set Variable node is overwritten by the second node's `foo: "second value"` output, but the "x" variable is preserved since the second node does not set a variable with the same name.

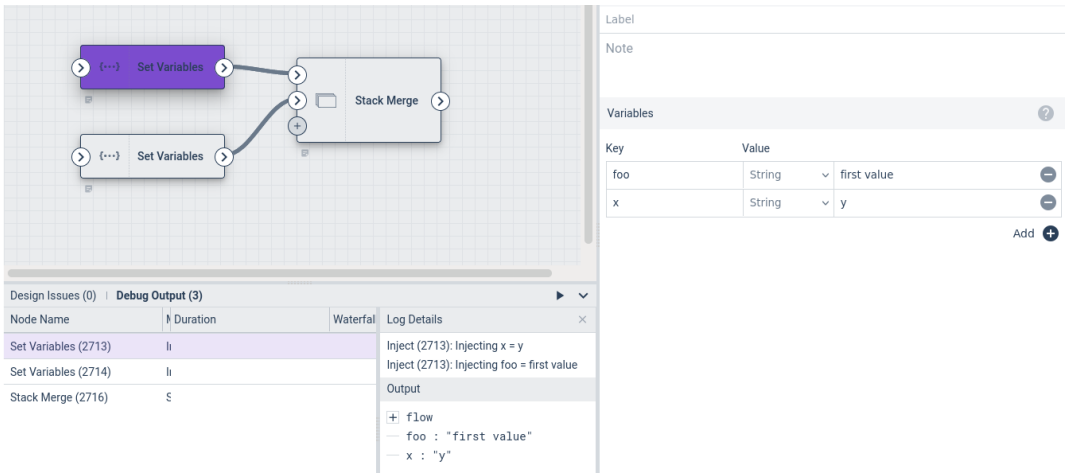


Figure 8.22: Stack merge inputs.

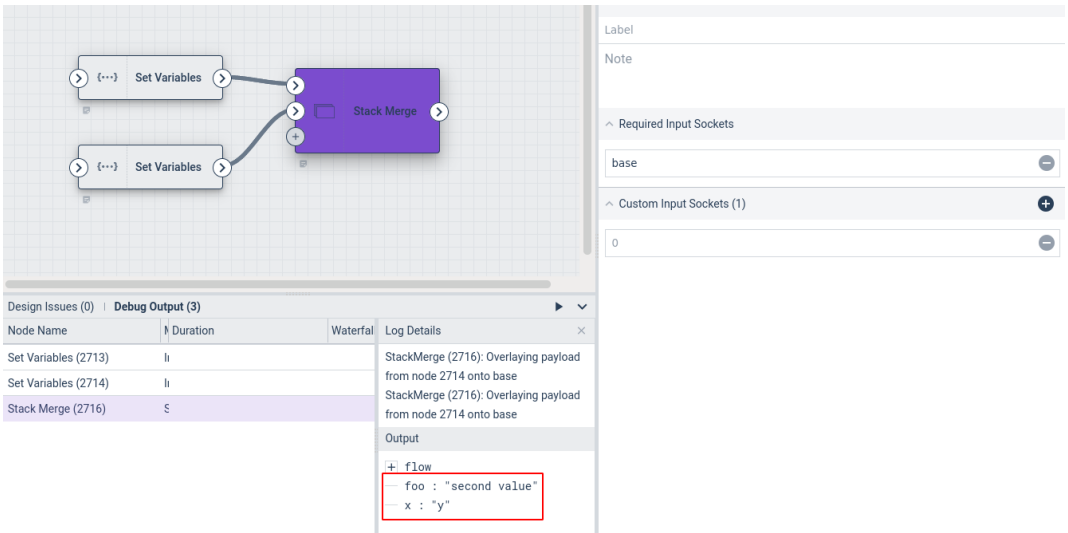


Figure 8.23: Stack merge output.

Email Node

The Email node sends emails from within a flow. It can attach items from the payload. Note that the user who owns the flow *must* configure an email server as described in Section 3.2.3 for the Email node to function properly!

- Sender, required: This is the address which will appear in the "From" header of the email.
- Recipients, required: The email will be sent to this address or addresses.
- Subject, required: The subject line of the email.
- Body, required: The body text of the email message. Enter a string manually, or select a variable containing suitable text. The Text Template node provides powerful tools for formatting text in the flow.
- Attachments: An optional array of items to add as attachments on the email.

The Email node makes a best-effort attempt at determining the appropriate file type on the attachment. Naming payload items with appropriate file extensions (e.g. appending `.pdf` to the output name of the PDF node) helps the Email node figure out the correct type, but it can also make some deductions based on the structure of the data.

JavaScript Node

The JavaScript node can execute JavaScript code in a flow, allowing complex logic and custom operations on the payload. The configuration of a JavaScript node consists of three items:

- Code: the actual JavaScript code to execute.
- Libraries: a set of libraries to load; these are key-value pairs, where the key is the name of the library (for user reference only) and the value is the content of the library.
- Outputs: a list of variables to be output from the script (see below).

Scripts can read from and write to the payload by accessing the `payload` variable, e.g. `payload.flow.Scheduled`. Note that variables *created* in the payload are only visible to downstream nodes if that variable is explicitly listed in the "Outputs" configuration field.

Combining the HTTP node and the JavaScript node makes it trivially easy to load JS libraries from the Internet. For instance, the flow in Figure 8.24 uses the HTTP node to fetch the Lodash JavaScript library[4], then the JavaScript node loads the HTTP response as a library and uses it to round a number. (Note that the Lodash library loads itself into a variable named `_`, so calling the round function from the Lodash library looks like `_.round(4.123)`)

PDF Node

The PDF node formats search results and other data into an attractive PDF document, which can then be sent to recipients using the Email node, the Slack node, the HTTP node, etc.

There are many configuration options on the PDF node:

- Title, required: the title of the PDF.
- Subtitle: an optional sub-title.
- Contents, required: select one or more items from the payload to be included in the PDF. Query results will be automatically formatted.
- Page Size: change the size of the pages in the PDF.

[4] https://raw.githubusercontent.com/lodash/lodash/4.17.15-npm/lodash.js

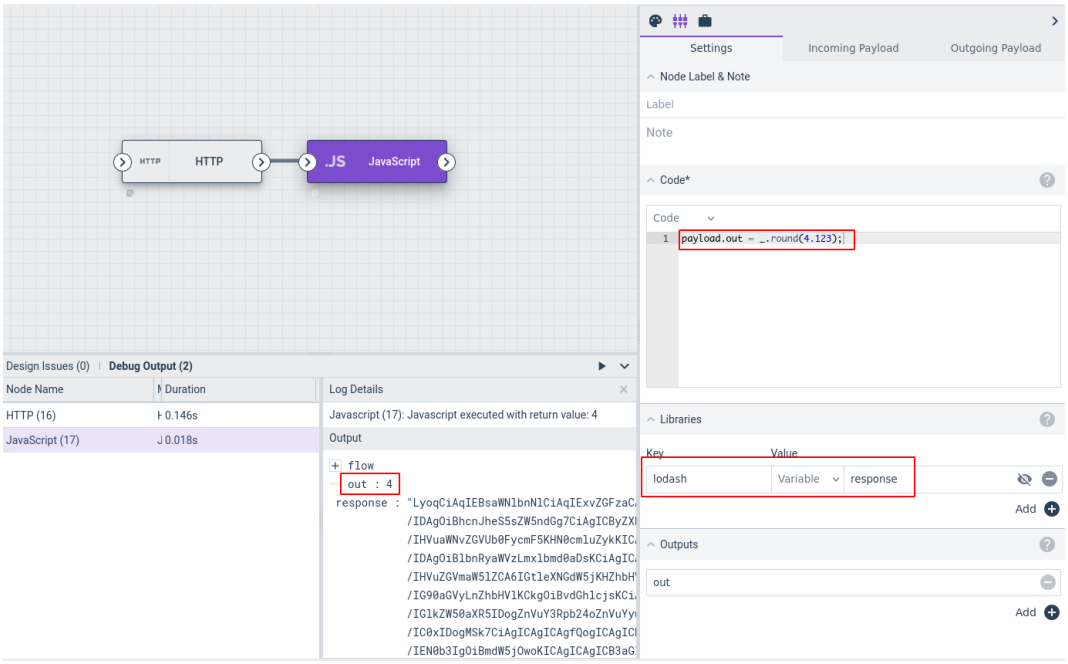


Figure 8.24: Loading JavaScript libraries.

- Include Flow Metadata Page: if set to true, the PDF will include a final page giving information about the execution of the flow.
- Password: if set, the PDF will be password-protected.
- Output Variable Name: sets the name for the output PDF in the payload.

The "Contents" field is perhaps the most critical. The node will attempt to format each item in this field as a section within the PDF document. It knows how to parse query results, meaning that a query using the table renderer will be inserted into the PDF as a proper table. Figure 8.25 shows an example flow in which the outputs of several queries, plus the output of a Text Template node, are used as the Contents of a PDF. Figure 8.26 shows an example of what such a PDF may look like.

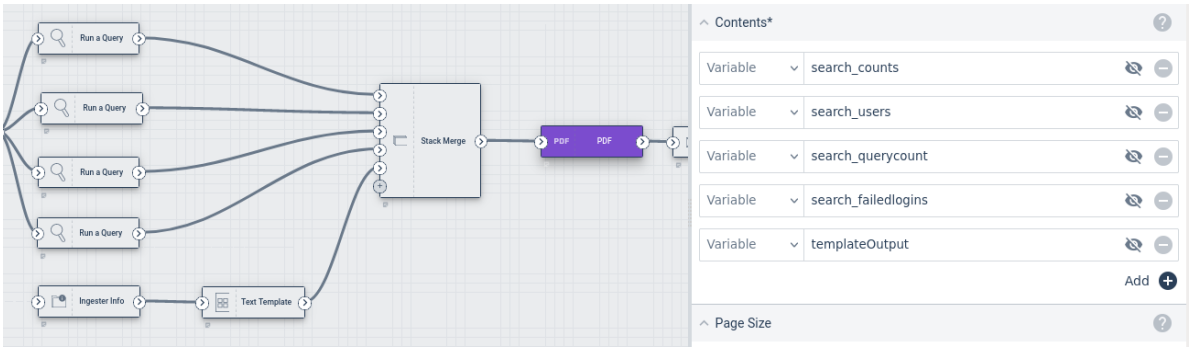
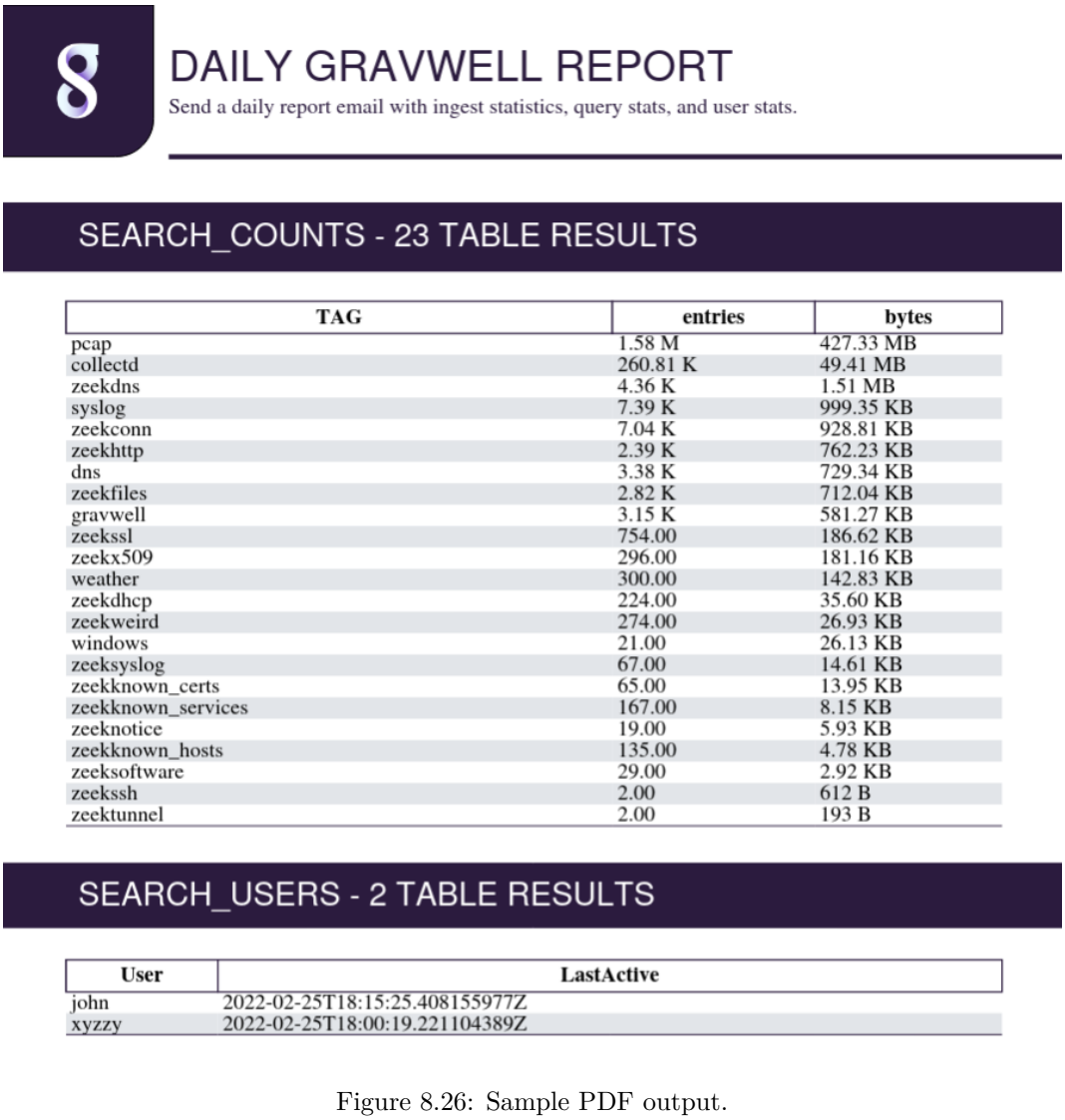


Figure 8.25: Including multiple query results in a PDF.

The "Output Variable Name" config option specifies the name to use for the output PDF in the payload. Using a name which ends in `.pdf` will help inform other nodes that the object is a PDF file, for instance when included as an attachment on the Email node.

Setting "Include Flow Metadata Page" to true causes the node to insert an additional page at the end of the



DAILY GRAVWELL REPORT

Send a daily report email with ingest statistics, query stats, and user stats.

SEARCH_COUNTS - 23 TABLE RESULTS

TAG	entries	bytes
pcap	1.58 M	427.33 MB
collectd	260.81 K	49.41 MB
zeekdns	4.36 K	1.51 MB
syslog	7.39 K	999.35 KB
zeekconn	7.04 K	928.81 KB
zeekhttp	2.39 K	762.23 KB
dns	3.38 K	729.34 KB
zeekfiles	2.82 K	712.04 KB
gravwell	3.15 K	581.27 KB
zeekssl	754.00	186.62 KB
zeekx509	296.00	181.16 KB
weather	300.00	142.83 KB
zeekdhcp	224.00	35.60 KB
zeekweird	274.00	26.93 KB
windows	21.00	26.13 KB
zeeksyslog	67.00	14.61 KB
zeekknown_certs	65.00	13.95 KB
zeekknown_services	167.00	8.15 KB
zeeknotice	19.00	5.93 KB
zeekknown_hosts	135.00	4.78 KB
zeeksoftware	29.00	2.92 KB
zeekssh	2.00	612 B
zeektunnel	2.00	193 B

SEARCH_USERS - 2 TABLE RESULTS

User	LastActive
john	2022-02-25T18:15:25.408155977Z
xyzzy	2022-02-25T18:00:19.221104389Z

Figure 8.26: Sample PDF output.

document containing additional information about the user who executed the flow, the status of the Gravwell cluster, and the current Gravwell license. It will also insert the full query string, the timeframe, and the execution time for any queries which were packaged into the PDF.

Slack File/Message Nodes

Two nodes can send to Slack: Slack Message and Slack File. As the names imply, one sends a text message to a Slack channel, while the other uploads a file. Both nodes have the following configuration options:

- Token, required: a Slack access token[5] for the desired server.
- Channel, required: the channel that will receive the message, *without* a preceding '#' character.
- Message: the message body; Markdown is supported.

The Slack Message node also has a "Verbatim Text" config, which is optional text to be attached to the message verbatim, without being parsed as Markdown. This is useful for showing the result of an HTTP request or other unformatted data.

The Slack File node also has a "File to Send" config, which is an item from the payload to be attached to the message. This works particularly well with the output of the PDF node.

Note that any user with read access to the flow will be able to extract the token, giving them the ability to write to the same Slack server. For this reason, we recommend that flows with Slack nodes should not be shared with other users.

Teams Node

The Teams node can send a message to a Microsoft Teams channel. The configuration options are:

- Webhook, required: an incoming webhook[6] URL for Microsoft Teams.
- Title: an optional title for the message.
- Message, required: the body of the message to send.

Note that any user with read access to the flow will be able to extract the webhook, giving them the ability to write to the same Teams channel. For this reason, we recommend that flows with Teams nodes should not be shared with other users.

[5] https://api.slack.com/authentication/token-types
[6] https://docs.microsoft.com/en-us/microsoftteams/platform/webhooks-and-connectors/what-are-webhooks-and-connectors

8.3.4 Hands-On Lab: Flows

This lab will demonstrate how to build a flow. The flow will send a notification whenever there have been failed login attempts to the Gravwell system. First, create a Gravwell webserver+indexer container:

```
docker run -d --rm --net gravnet -p 8080:80 --name gravwell gravwell:base
```

Now log into the web GUI (http://localhost:8080). After that, open a new *private* browser window (so the existing login cookie isn't used) and enter some invalid login credentials, e.g. username "admin" with password "admin". This will generate a login failure message which we can check using the following query:

```
tag=gravwell syslog Message=="Authentication failure" user
| stats count by user
| table user count
```

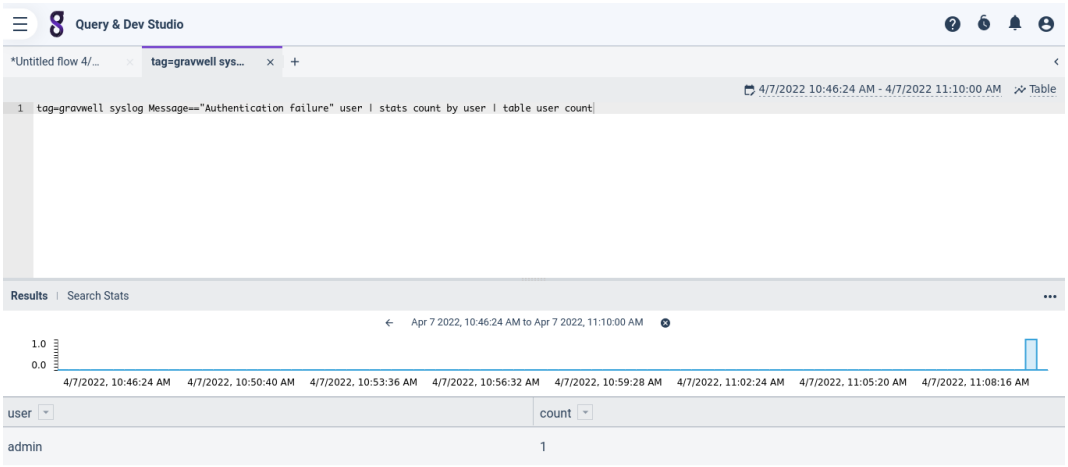


Figure 8.27: Query showing failed login attempts.

We will use that query as the basis for a new flow. Select "Flows" from the "Automation & Flows" section in the main menu, then click the "+" icon in the upper right to create a new flow. Drag the following nodes from the node palette out to the canvas:

- Run a Query
- If
- Gravwell Notification

Wire them up as seen in Figure 8.28

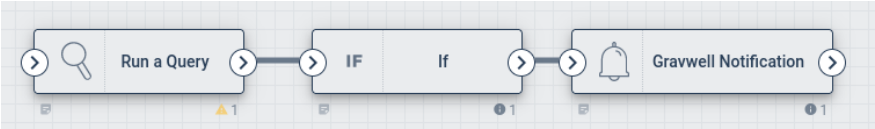


Figure 8.28: Node layout for lab.

Each node now needs to be configured. Begin with the Run a Query node; click it, then paste the Gravwell query above into the "Query String" config field. Note how the "Search Timeframe" defaults to the last hour, and "Output Variable Name" defaults to "search". This means the payload output by the Run a Query node will contain an item named "search" with information about the query which was executed.

The If node should be configured to check `search.Count > 0`. This means that execution will continue if there were any results from the search.

Finally, set a message in the Gravwell Notification node's Message field, something like "Failed logins! Go check!".

Once all three nodes are configured, the Design Issues tab of the console should be empty. If there are issues reported, check your configuration on the associated node.

To test the flow, click the debug icon in the toolbar. You should soon see a notification appear as in Figure 8.29.

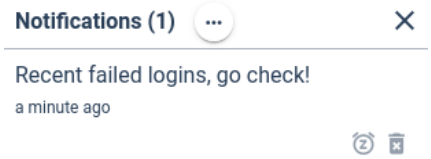


Figure 8.29: Example notification from flow.

If no notification appears, double-check that the query given above actually returns results when run over the last hour, or try generating more failed login attempts.

Once the flow is working, you can prepare the flow for deployment. In the Run Query node's config, change the Search Timeframe dropdown from "ISO Duration" to "Variable". The value should change to the default `flow.Interval`. Then go to the Info & Scheduling tab, set the schedule to "* * * * *" to make it run every minute, toggle the "Disable scheduling" slider, and hit Save. These two changes will make the flow run automatically every minute, and cause the query to run only over the last minute (this is the effect of setting the duration to `flow.Interval`).

Test these changes by periodically creating failed login events. You should see a notification pop up after a failed login. Delete the notification; it should not come back again until you do another failed login.

To clean up after the experiment, run:

```
docker kill $(docker ps -a -q)
```

8.4 Search Scripting and Orchestration

Please note that for most use cases, Gravwell flows are a much easier solution than writing a script.

Scripts provide an additional layer of power beyond scheduled searches. A script can execute multiple searches, filter and enrich the results, then re-ingest the resulting entries under a different tag, send alerting emails, or do HTTP requests automatically.

Scripts can be run on a schedule, just like scheduled searches, or they can be run manually with the command line client. This section will first discuss how scripts are written, then delve into examples of how they can be run and used.

A script can be scheduled in exactly the same way as a scheduled search by simply changing the dropdown in the scheduled search creation form from "Search query" to "Anko script" and pasting the script into the form, as shown in Figure 8.30.

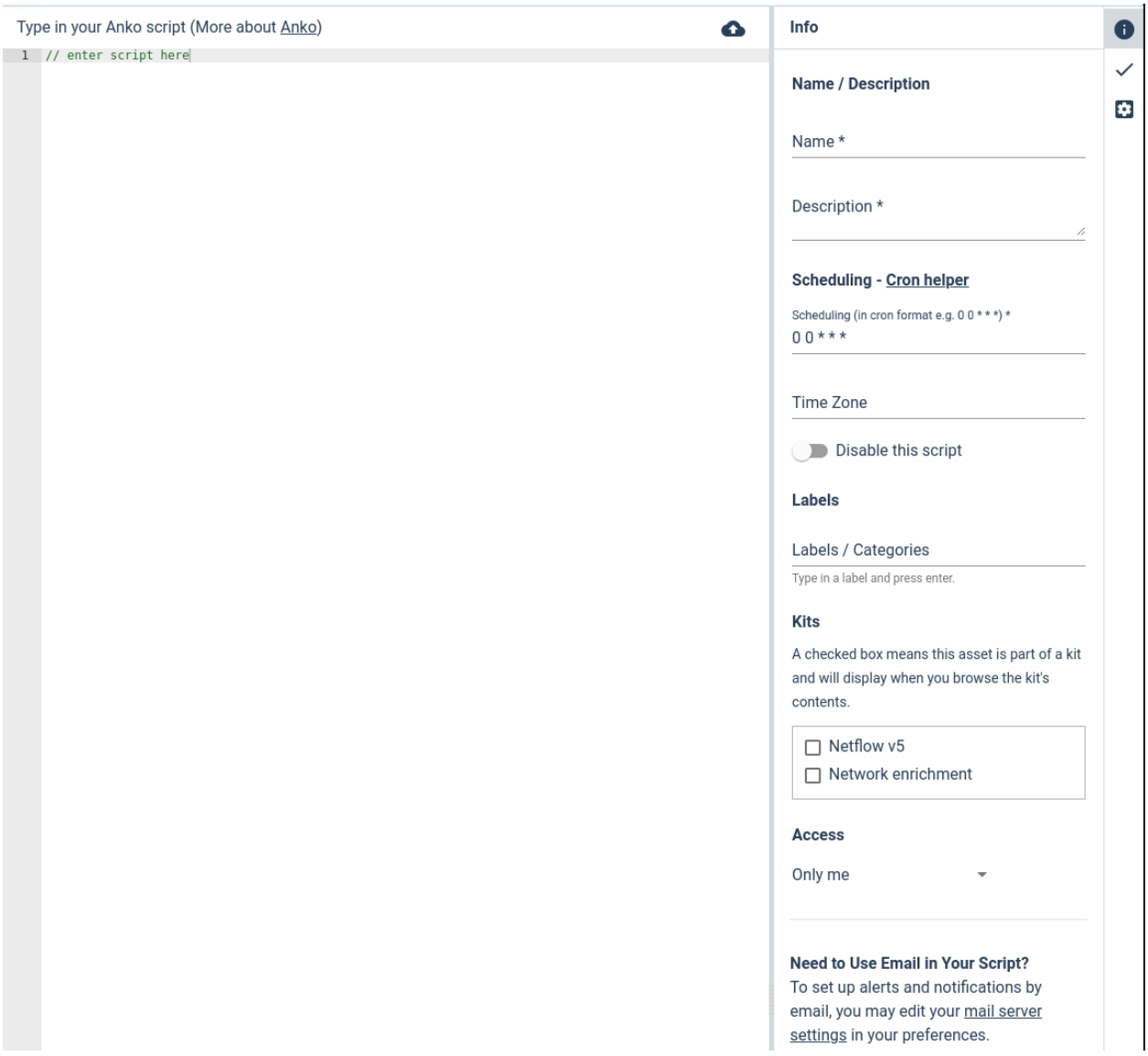


Figure 8.30: Creating a new scheduled script

The rest of this section will describe what scripts look like and how they are written.

8.4.1 The Anko Scripting Language

Gravwell scripts are written in the Anko scripting language (see http://github.com/mattn/anko). Anko is an interpreted, dynamically-typed language whose syntax resembles Go. This section gives a brief overview of the language.

Comments

Anko comments can be prefixed with either the hash character or a pair of slashes, or they can be wrapped in C-style comment markers:

```
// This is a valid Anko comment
# This is also a valid comment
/* This is a comment, too */
```

Declaring variables

In Anko, variables can be created and modified with the same syntax:

```
// This creates the variable
x = 1
// And this updates it
x = x + 1
```

The 'var' keyword can be used to indicate that a *new* variable *must* be allocated at the current scope:

```
// Declare a variable
var x = 1

if true {
    // Declare a new variable at the inside scope
    var x = 2
}
println(x)    // prints "1"
```

If the 'var' keyword was not used, the `println` call would have printed '2'.

Slices

Anko implements slices (arrays) in a similar way to Go, but with more implicit operations allowed:

```
a = [1, 2, 3]
a += 4        // append
println(a)    // prints [1 2 3 4]
println(a[1])    // prints 2
```

Anko's default slice type is `[]interface{}`, which means a slice can contain many different types. Note that slices are "expanded" when appended:

```
a = [1, 2.2, "foo"]
a += [3, 4]
a += [ [5, 6] ]
println(a)    // prints [1 2.2 foo 3 4 [5 6]]
```

Maps

Anko maps are maps of `interface{}` to `interface{}`, meaning a given map can mix key and value types:

```
m = { "foo": 2 }    // initializes a map with a single entry, "foo" -> 2
m[3] = 7        // map key 3 to value 7
println(m[3])    // prints "7"
println(m["foo"]) // prints "2"
```

If-Else Statements

If-ElseIf-Else statements work exactly the same in Anko as in Go:

```
if x > 10 || y {
    count1++
} else if x == 3 && !y {
    count2++
} else {
    count3++
}
```

Switch Statements

Anko provides switch statements similar to Go. Note that there is no "fallthrough" keyword.

```
switch val {
case "foo":
    count1++
case 3:
    count2++
default:
    notfound++
}
```

For Loops

Anko supports traditional for loops:

```
for i = 0; i < 10; i++ {
  println(i)
}
```

It also can also iterate over the contents of a slice:

```
a = ["foo", 3, 8.3]
for i in a {
  println(i) // will print "foo", 3, 8.3
}
```

The 'range' built-in function is convenient for iteration:

```
// this will print 0 through 9
for i in range(0, 10) {
  println(i)
}

// this will print 0, 2, 4, 6, 8
for i in range(0, 10, 2) {
```

```
  println(i)
}
```

Function Declaration

Anko functions resemble Go functions, but they do not declare any return types. Arguments are passed by value, not by reference:

```
func foo(x) {
  return x++
}

var a = 1
foo(a)
println(a)    // prints "1"
a = foo(a)
println(a)    // prints "2"
```

8.4.2 Available Libraries

A selection of Go libraries have some functionality exported to Anko for use in scripts. Check the Gravwell online documentation for a full list. Some examples are:

- “bytes”
- “encoding/json”
- “errors”
- “math”
- “math/big”
- “math/rand”
- “net/url”
- “path”
- “path/filepath”
- “regexp”
- “sort”
- “strings”
- “time”
- “io”
- “flag”
- “io/ioutil”
- “encoding/csv”
- “crypto/md5”
- “crypto/sha1”
- “crypto/sha256”
- “crypto/sha512”
- “net”

- "net/http"
- "github.com/ziutek/telnet"
- "google/uuid" (github.com/google/uuid)

Libraries must be imported using the import function before use:

```
var md5 = import("crypto/md5")
fooSum = md5.Sum("foo")
```

8.4.3 Gravwell Anko Functions

Gravwell has extended the Anko language with additional functions specifically suited to writing orchestration scripts. This section describes those functions. The functions are listed below in the format `functionName(<functionArgs>) <returnValues>`. Functions which return more than one argument have the return values wrapped in parentheses.

Resources and persistent data functions

"Persistent maps" are a convenience offered for scheduled searches. Values set in a persistent map will be available for reading in subsequent runs of the scheduled search. If the named persistent map does not exist, it will be created. Persistent maps do nothing when the script is run at the Gravwell CLI.

- `getResource(name) ([]byte, error)` - returns the contents of the specified resource as a slice of bytes, while the error is any error encountered while fetching the resource.
- `setResource(name, value) error` - creates (if necessary) and updates a resource named name with the contents of value, returning an error if one arises.
- `setPersistentMap(mapname, key, value)` - stores a key-value pair in a map which will persist between executions of a scheduled script.
- `getPersistentMap(mapname, key) value` - returns the value associated with the given key from the named persistent map.
- `delPersistentMap(mapname, key)` - deletes the specified key/value pair from the given map.

Search entry manipulation functions

These functions get, set, and delete enumerated values on a single entry (as returned by the `getEntries` function described later)

- `setEntryEnum(ent, key, value)` - sets an enumerated value on the specified entry.
- `getEntryEnum(ent, key) (value, error)` - reads an enumerated value from the specified entry.
- `delEntryEnum(ent, key)` - deletes the specified enumerated value from the given entry.

General utility functions

- `len(val) int` - returns the length of val, which can be a string, slice, etc.
- `toIP(string) IP` - converts string to an IP, suitable for comparing against IPs generated by e.g. the packet module.
- `toMAC(string) MAC` - converts string to a MAC address.
- `toString(val) string` - converts val to a string.
- `toInt(val) int64` - converts val to an integer if possible. Returns 0 if no conversion is possible.
- `toFloat(val) float64` - converts val to a floating point number if possible. Returns 0.0 if no conversion is possible.
- `toBool(val) bool` - attempts to convert val to a boolean. Returns false if no conversion is possible. Non-zero numbers and the strings "y", "yes", and "true" will return true.
- `typeOf(val) type` - returns the type of val as a string, e.g. "string", "bool".

Search management functions

Due to the way Gravwell's search system works, some of the functions in this section return Search structs (written as `"search"` in the parameters) while others return search IDs (written as `"searchID"` in the parameters). Each Search struct contains a search ID, which can be accessed as `search.ID`.

Search structs are used to actively read entries from a search, while search IDs tend to refer to inactive searches to which we may attach or otherwise manage.

- `startBackgroundSearch(query, start, end) (search, err)` - creates a backgrounded search with the given query string, executed over the time range specified by 'start' and 'end'. The return value is a Search struct. These time values should be specified using the time library; see the examples for a demonstration.
- `startSearch(query, start, end) (search, err)` - acts exactly like `startBackgroundSearch`, but does not background the search.
- `detachSearch(search)` - detaches the given search (a Search struct). This will allow non-backgrounded searches to be automatically cleaned up and should be called whenever you're done with a search.
- `attachSearch(searchID) (search, error)` - attaches to the search with the given ID and returns a Search struct which can be used to read entries etc.
- `getSearchStatus(searchID) (string, error)` - returns the status of the specified search, which can be "ACTIVE", "ATTACHED", "DORMANT", "SAVED", "SAVING", or "UNKNOWN".
- `getAvailableEntryCount(search) (uint64, bool, error)` - returns the number of entries that can be read from the given search, a boolean specifying if the search is complete, and an error if anything went wrong.
- `getEntries(search, start, end) ([]SearchEntry, error)` - pulls the specified entries from the given search. The bounds for start and end can be found with the `getAvailableEntryCount` function.
- `isSearchFinished(search) (bool, error)` - returns true if the given search is complete
- `executeSearch(query, start, end) ([]SearchEntry, error)` - starts a search, waits for it to complete, retrieves up to ten thousand entries, *detaches* from search and returns the entries.
- `deleteSearch(searchID) error` - deletes the search with the specified ID
- `backgroundSearch(searchID) error` - sends the specified search to the background; this is useful for "keeping" a search for later manual inspection.
- `downloadSearch(searchID, format, start, end) ([]byte, error)` - downloads the given search as if a user had clicked the 'Download' button in the web UI. `format` should be a string containing either "json", "csv", "text", "pcap", or "lookupdata" as appropriate. `start` and `end` are time values.
- `getDownloadHandle(searchID, format, start, end) (io.Reader, error)` - returns a streaming handle to the results of the given search as if the user had clicked the 'Download' button in the web UI. The handle returned is suitable for use with the HTTP library functions shown later in this document.

Search Data Type

When executing a search via the `startSearch` or `startBackgroundSearch` functions the *search* data type is returned. The search data type contains the following members:

- `ID` - A string containing the search ID. Use this member for other functions like `getSearchStatus` and `attachSearch`.
- `RenderMod` - A string indicating the renderer attached to the search. It may be something like raw, text, table, chart, or fdg.
- `SearchString` - A string containing the search string passed in during the request
- `SearchStart` - A string containing the start timestamp for the search
- `SearchEnd` - A string containing the end timestamp for the search
- `Background` - A boolean indicating whether the search was started as a background search
- `Name` - An optional string with a search name.

Transmitting alerts or search results

The scripting system provides several methods for transmitting script results to external systems.

The following functions provide basic HTTP functionality:

- `httpGet(url) (string, error)` - performs an HTTP GET request on the given URL, returning the response body as a string.
- `httpPost(url, contentType, data) (response, error)` - performs an HTTP POST request to the given URL with the specified content type (e.g. "application/json") and the given data as the POST body.

More elaborate HTTP operations are possible with the "net/http" library. See the package documentation in the online anko documentation[7] for a description of what is available.

If the user has configured their personal email settings within Gravwell, the email function is a very simple way to send an email:

- `email(from, to, subject, message) error` - sends an email via SMTP. The from field is simply a string, while the to field should be a slice of strings containing email addresses. The subject and message fields are also strings which should contain the subject line and body of the email.

Creating and ingesting entries

It is possible to ingest new entries into the indexers from within a script using the following functions:

- `newEntry(time, data) Entry` - returns a new entry with the given timestamp (a `time.Time`, as from `time.Now()`) and `data` (frequently a string).
- `ingestEntries(entries, tag) error` - ingests the given slice of entries (`[]Entry`) with the specified tag string.

The entries returned by the `getEntries` function can be modified if desired and re-ingested via `ingestEntries`, or new entries can be created from scratch. For example, to re-ingest some entries from a previous search into the tag "newtag":

```
# Get the first 100 entries from the search
ents, _ = getEntries(mySearch, 0, 100)
ingestEntries(ents, "newtag")
```

To ingest new entries based on some other condition:

```
if condition == true {
    ents = make([]Entry)
    ents += newEntry(time.Now(), "Script condition triggered")
    ingestEntries(ents, "results")
}
```

8.4.4 Example Scripts

This script creates a backgrounded search that finds which IPs have communicated with Cloudflare's 1.1.1.1 DNS service over the last day. If no results are found, the search is deleted, but if there are results the search will remain for later perusal by the user in the 'Persistent Searches' screen of the GUI.

```
# Import the time library
var time = import("time")
# Define start and end times for the search
start = time.Now().Add(-24 * time.Hour)
end = time.Now()
# Launch the search
s, err = startSearch("tag=netflow netflow Dst==1.1.1.1 Src | unique Src | table Src",
↪  start, end)
if err != nil {
```

[7] https://docs.gravwell.io/#!scripting/scriptingsearch.md

```
    return err
}
# Wait until the search is finished
for {
    f, err = isSearchFinished(s)
    if err != nil {
        return err
    }
    if f {
        break
    }
    time.Sleep(1 * time.Second)
}
# Find out how many entries were returned
c, _, err = getAvailableEntryCount(s)
if err != nil {
    return err
}
# If no entries returned, delete the search
# Otherwise, background it
if c == 0 {
    deleteSearch(s.ID)
} else {
    err = backgroundSearch(s.ID)
    if err != nil {
        return err
    }
}
# Always detach from the search at the end of execution
detachSearch(s)
```

8.4.5 Developing & Testing Scripts

Because the same script can be executed as either a scheduled search or manually with the CLI client, scripts are usually tested/developed using the CLI client and then copied into a scheduled search later. Scripts run with the CLI client can display printed output, which is ignored when run on a schedule; this is particularly useful when debugging a script.

A script can be executed manually from the client in the following way:

```
$ gravwell -s <server> script
script file path> /path/to/script.ank
```

A convenient way to run a script repeatedly is to use the "watch" modifier. The client will execute the script, then prompt to re-run. If the script file is modified between runs, the next execution will use the updated script. Here, a script that hashes a string with MD5 is modified to hash using SHA1:

```
$ gravwell -s <server> watch script
script file path>  /tmp/hash.ank
[172 189 24 219 76 194 248 92 237 239 101 79 204 196 164 216]
Hit [enter] to re-run, or [q][enter] to cancel

[11 238 199 181 234 63 15 219 201 93 13 212 127 60 91 194 117 218 138 51]
Hit [enter] to re-run, or [q][enter] to cancel
```

8.4.6 Hands-on Lab: Scripting

This lab will demonstrate how to send email alerts from a script. This lab will test the script using the Gravwell CLI client, then schedule it for automated execution.

The script will operate on Netflow records and send an alert whenever traffic is seen originating from the 7.0.0.0/8 subnet, which in this example stands in for a "known bad" subnet or subnets. In an actual deployment, one might keep a list of known-bad subnets in a resource lookup table (which can be automatically updated via another script!) and compare flow records against that lookup table instead, but for simplicity we will use a single hard-coded subnet.

First, launch a Gravwell webserver+indexer container:

```
docker run --rm --net gravnet -p 8080:80 -d --name gravwell gravwell:base
```

Log in, go to the Account settings page and select the Email Server tab, then populate your email server configuration and test it (Figure 8.31).

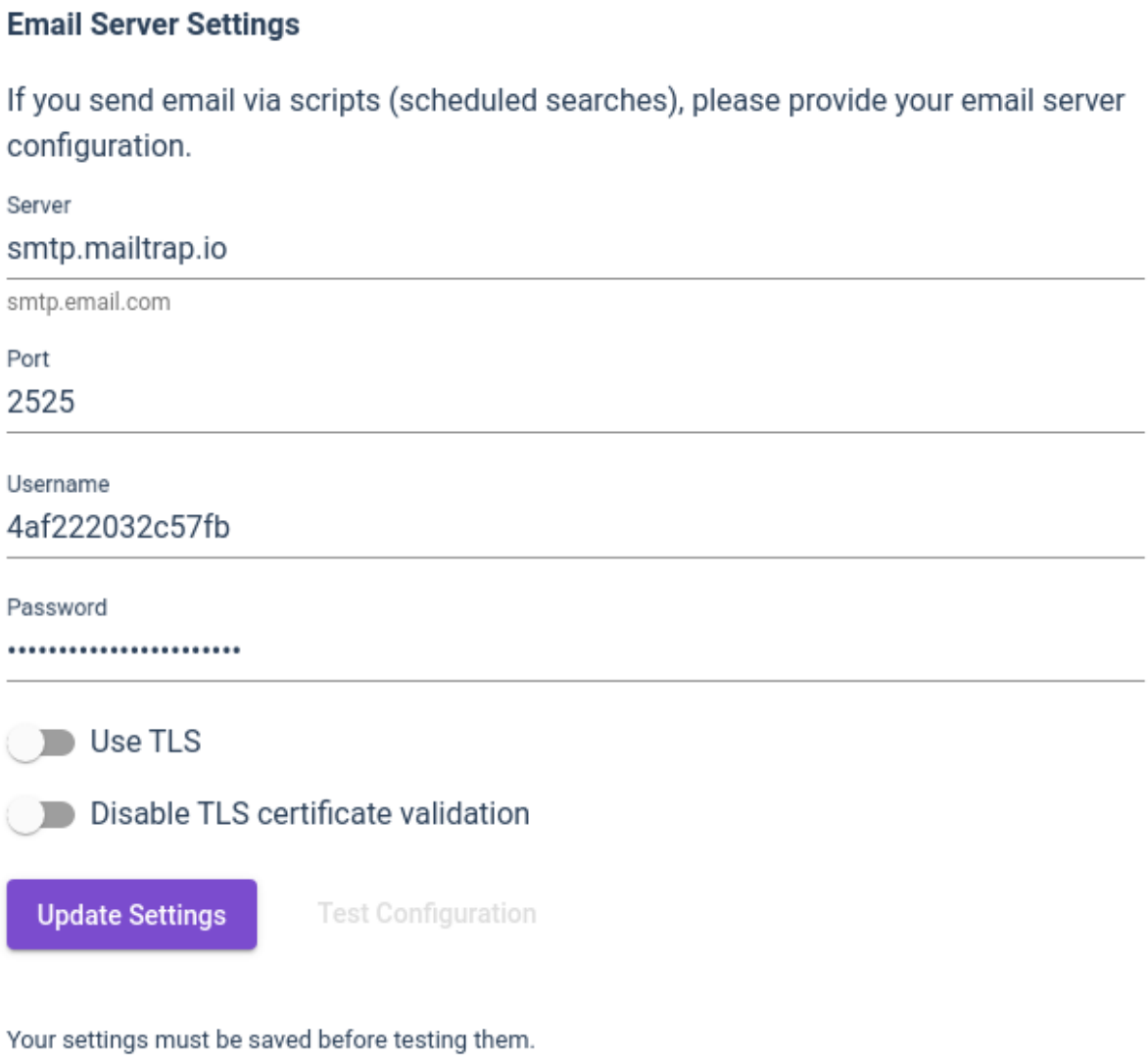


Figure 8.31: Configuring email server

Next, start the ingester container running the Netflow ingester:

```
docker run --rm -d --net gravnet --name ingesters \
-e GRAVWELL_CLEARTEXT_TARGETS=gravwell:4023 gravwell:ingesters \
/opt/gravwell/bin/gravwell_netflow_capture
```

The Netflow ingester is pre-configured to listen on port 2055 for incoming Netflow v5 records.

Now, we use another Docker container to generate Netflow records and send them to the ingester:

```
docker run -it --net gravnet --rm \
networkstatic/nflow-generator -t ingesters -p 2055
```

The Netflow generator will run indefinitely, generating flow records, until killed. Let it run at least until the following query shows at least one result:

```
tag=netflow netflow Src | subnet Src /8 as sub | ip sub == 7.0.0.0 | table
```

This query will form the basis of our script. It extracts the source IP address from the Netflow records, converts that to a /8 subnet and stores the result in an enumerated value named "sub", then compares "sub" to 7.0.0.0.

Open the file named `email-netflow.ank` on the host system and paste the following into it (or use the version in the training package):

```
var time = import("time")
# SET THESE VARIABLES
from = "MY.ADDRESS@EXAMPLE.COM"
to = [ "RECIPIENT@EXAMPLE.COM" ]
my_name = "HANK"
# Do the search over the last day
end = time.Now()
start = end.Add(-24 * time.Hour)
query = `tag=netflow netflow Src | subnet Src /8 as sub | ip sub == 7.0.0.0 | table`
s, err = startSearch(query, start, end)
if err != nil {
        return err
}
# Wait for the search to complete
for {
        f, err = isSearchFinished(s)
        if err != nil {
                return err
        }
        if f {
                break
        }
        time.Sleep(1 * time.Second)
}
# Figure out how many results there were
c, _, err = getAvailableEntryCount(s)
if err != nil {
        return err
}
# clean up the search, we only care about how many entries there were
detachSearch(s)

# If there was more than 0 entries, send an email
```

```
if c > 0 {
        summary = "There were " + c + " flows originating from the banned subnet!"
        email(from, to, my_name + " alert!", summary)
}
```

Modify the `'from'`, `'to'`, and `'my_name'` variables at the top of the script; 'from' should be your email address, 'to' is a list of email recipients (this should probably be your email address again), and 'my_name' is your own name.

Now copy the file to the Gravwell container and run a shell:

```
docker cp email-netflow.ank gravwell:/tmp/
docker exec -it gravwell /bin/sh
```

Within that shell, run the Gravwell CLI client and execute the script:

```
$ gravwell -insecure-no-https script
Username:  admin
Password:  changeme
script file path>  /tmp/email-netflow.ank
```

If all goes well, an email alert should arrive soon!

The script can be trivially run on a schedule by simply pasting it into the New Scheduled Search dialog, as shown in Figure 8.32.

Note that this will send an email every minute, so either delete the scheduled script once it's been verified to work, or shut down the entire experiment when satisfied:

```
docker kill $(docker ps -a -q)
```

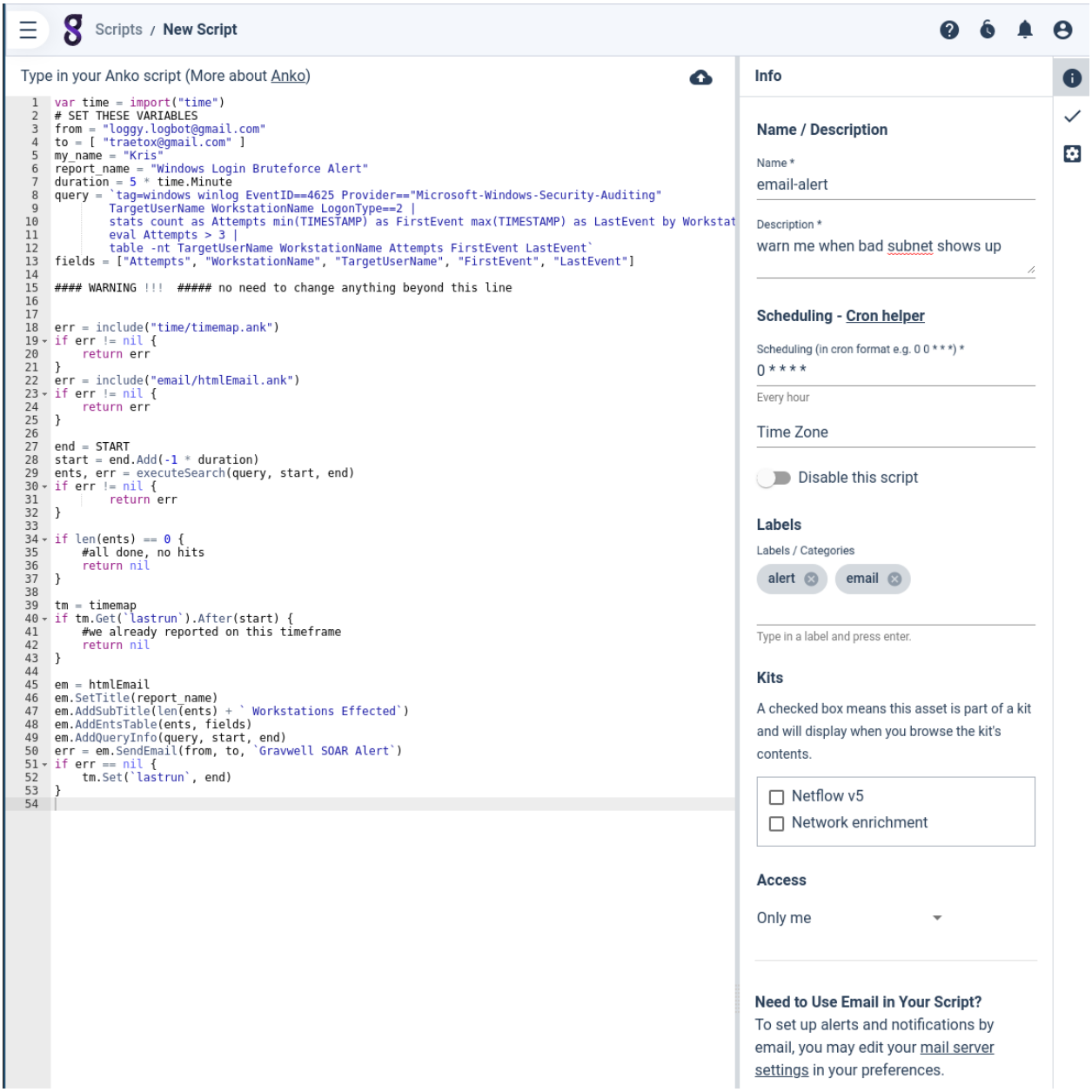


Figure 8.32: Creating scheduled script

Chapter 9

Gravwell Kits

Gravwell Kits act like a sort of 'app store' from which Gravwell users can install kits to gain out-of-box capabilities for common technologies. As an example, let's take an organization that uses Zeek for firewall/IDS logging on their network. These logs are ingested by Gravwell and easily searchable, but the users must issue their own queries to explore and analyze that data. This requires knowledge and experience with both Gravwell and Zeek in order to create useful dashboards, automation, data enrichments, or otherwise get value out of the data using Gravwell.

With Gravwell Kits, that organization can install the pre-built and signed kit for Zeek. This comes with:

- Commonly-used dashboards for quick situational awareness and heads-up monitoring of Zeek activity.
- Scheduled searches which look for statistical anomalies within the Zeek data and alert when any are discovered.
- Investigative dashboards that search Zeek data for items of interest such as IP addresses found in other logs or data within Gravwell.
- Search Templates for easily running common queries.
- Threat intelligence resources that search Zeek events for known-malicious activity.
- And more!

Creating Gravwell kits isn't very different than the hands-on activity we have been conducting to date. Building dashboards, setting up automated searches, loading data enrichment resources–all of these things are what goes into developing a cohesive kit. These Gravwell resources are bundled up, digitally signed, and distributed via Gravwell infrastructure to users. Because these kits can include Gravwell scripts (which are Turing-complete programs), the Gravwell team assesses and digitally signs all kits approved for distribution.

This chapter shows how to browse, install, and explore kits. It also discusses the kit building process, which allows you to package things you build to share with friends and colleagues.

The 'Kits' section of the Gravwell web UI is the centralized place to work with kits. You can find the Kits page in the main menu, as shown in Figure 9.1.

9.1 What's in a Kit?

There are many components which make up a kit. First, there are the contents of the kit, which fall into 2 categories:

- Items: Regular Gravwell components such as dashboards, scheduled searches, macros, actionables, etc.
- Configuration Macros: These are specialized macros which the kit uses to configure itself, which can allow greater flexibility in e.g. choices of tags used. For instance, rather than using tag=netflow in all queries, a Netflow kit can say `tag=$NETFLOW_KIT_TAG`, then define a configuration macro named

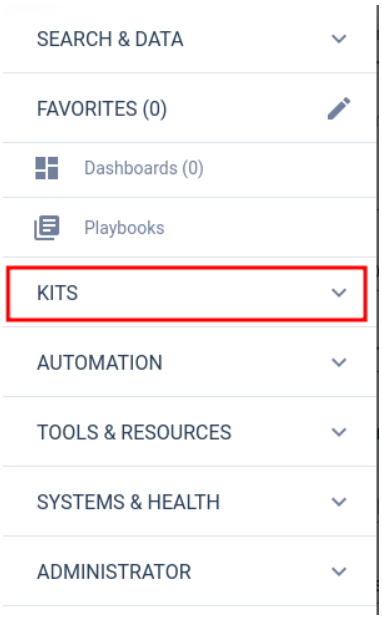


Figure 9.1: Kits menu item

`NETFLOW_KIT_TAG`. At installation time, the kit prompts the user for which tag or tags contain Netflow records.

There are a few other things which help identify a kit that are useful to keep in mind:

- ID: Identifies the kit. Gravwell uses namespaces similar to Android applications, e.g. "io.gravwell.netflow".
- Version: Kits may be updated over time, and the version number tracks this so Gravwell can automatically notify of new kit versions.
- Name: A user-friendly name for the kit, e.g. "Netflow v5".
- Description: A detailed description of what the kit does.
- MinVersion/MaxVersion: Some kits require specific Gravwell features; to ensure those features are available, these fields specify which Gravwell versions are compatible with the kit.
- Dependencies: Kits can depend on other kits, like packages in a Linux distribution. Gravwell's Netflow v5 kit depends on the Network Enrichment kit, for example. Dependencies are automatically installed along with the kit.

9.1.1 Dependencies

: A kit may have *dependencies* defined. Dependencies are other kits which the kit requires for proper functionality. For example, many kits depend on the Network Enrichment kit, which provides some baseline resources for enriching network data, such as a GeoIP database. Dependencies are installed automatically when you deploy a kit, provided the dependency exists on the Gravwell kit server.

9.2 Browsing and Installing Kits

On a fresh Gravwell cluster, no kits are installed. Opening the 'Kits' section will present an empty page (Figure 9.2). If you click 'Manage Kits', you will be taken directly to the list of kits on the Gravwell kit server, as shown in Figure 9.3.

You can click the kit details button to learn more about a given kit. Once you've decided on a kit to install, click the install kit button. The system will download the kit, then pop up a wizard for installation. In Figure 9.4, we have selected the IPFIX kit for installation. The first page shows a list of items contained in

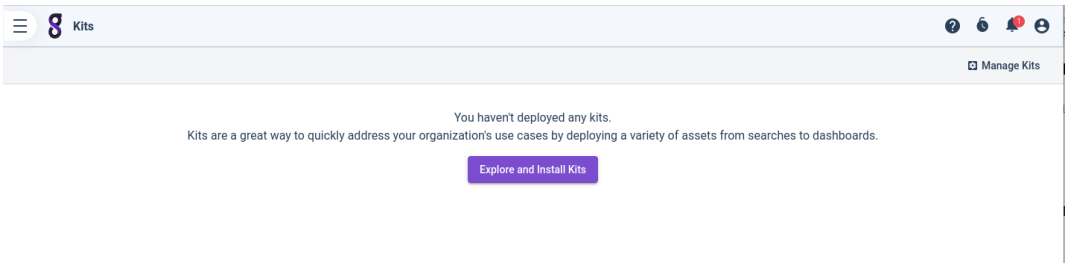


Figure 9.2: Kits page on a new Gravwell installation

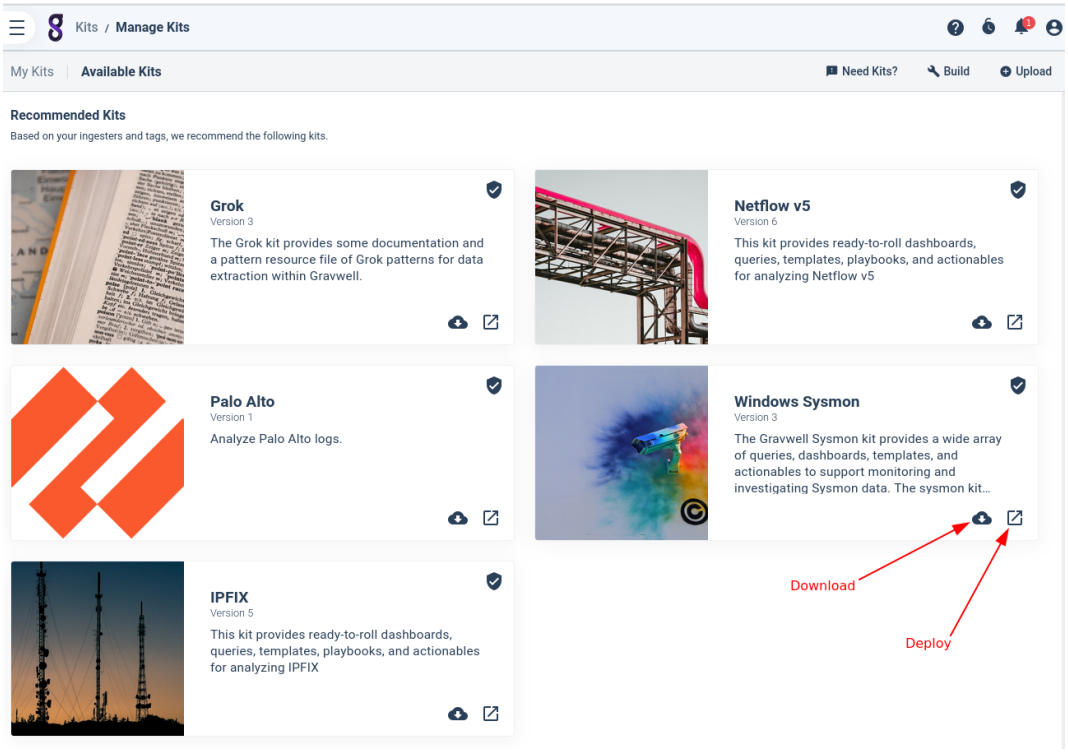


Figure 9.3: Browsing available kits

the kit; after reviewing the contents, click the checkbox and select the Next button. The wizard will then display any licenses packaged with the kit, as seen in Figure 9.5. Note that if there are multiple licenses, you will have to select each one individually from the list on the left and click each checkbox before continuing.

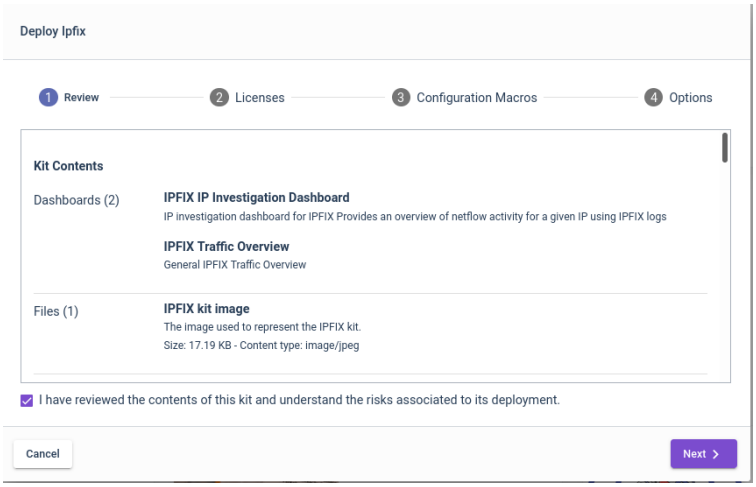


Figure 9.4: Installing IPFIX kit, step 1

Figure 9.5: Installing IPFIX kit, step 2

Next, the wizard will prompt for Configuration Macros, if any are defined by the kit. Figure 9.6 shows this step. A configuration macro allows install-time configuration of the queries which are shipped by the kit; these will typically include a default value but also provide a description to help you figure out what to enter. In the screenshot, it needs to know which tag contains IPFIX records; because we intend to use the `ipfix` tag, we can leave the default value alone.

The final page of the wizard (Figure 9.7 prompts for additional options. "Override Existing Items", if checked, will overwrite any conflicting objects which may already exist on the system–for instance, if you have created a resource named "foo", but the kit will also create a resource named "foo". The "Group Access" dropdown allows you to optionally select a group which can see the contents of the kit. Admin users will also have the option to install the kit globally, meaning all users can see it.

When you click the "Deploy" button, the kit and any dependencies will be installed. The GUI will display kit installation status as it goes. It may take some time, but eventually the installation will complete as shown in Figure 9.8.

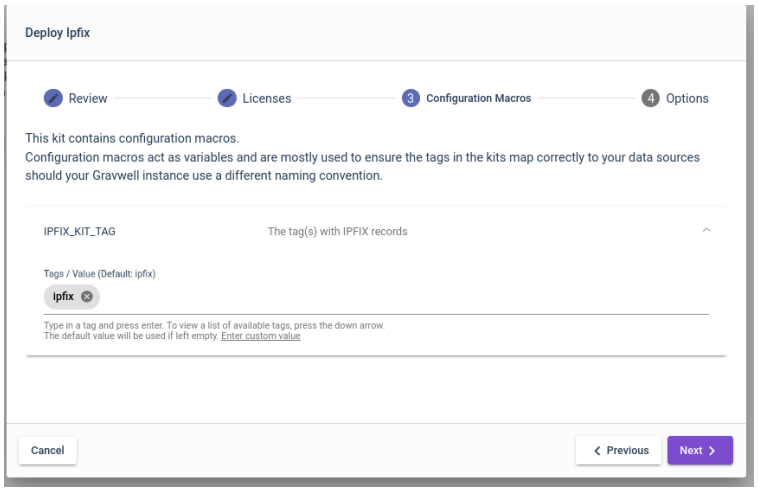


Figure 9.6: Installing IPFIX kit, step 3

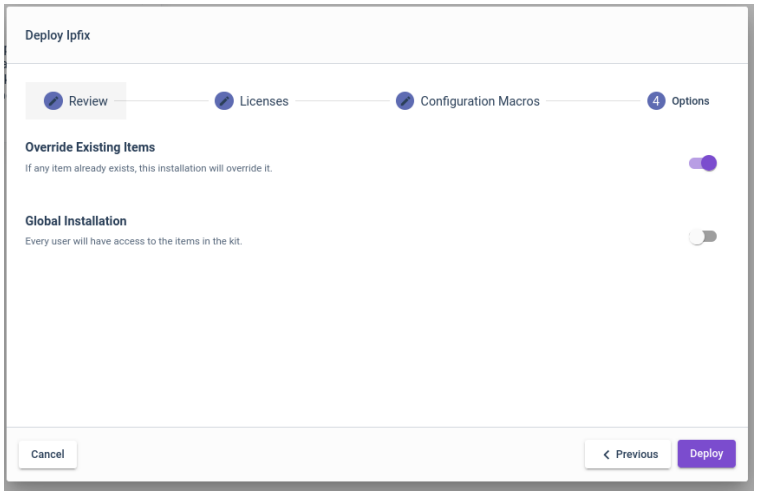


Figure 9.7: Installing IPFIX kit, step 4

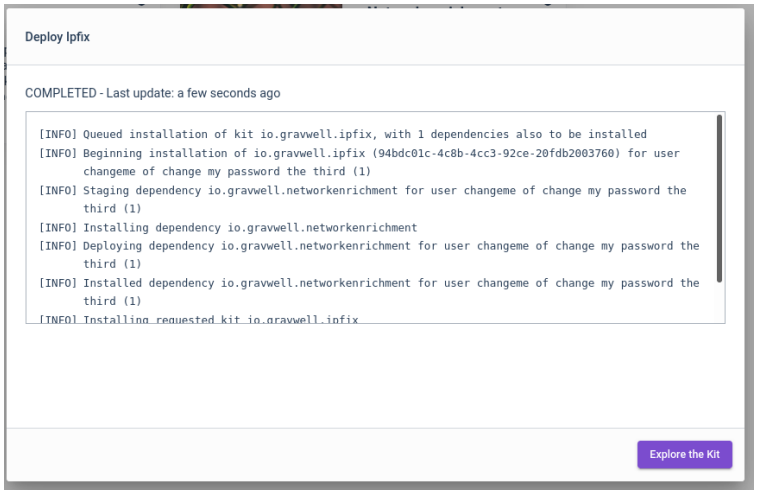


Figure 9.8: Kit deployment status

9.2.1 Exploring a Kit

If you click 'Explore the Kit' on the kit installation dialog (Figure 9.8, you'll be taken to a page outlining the contents of the kit, as seen in Figure 9.9.

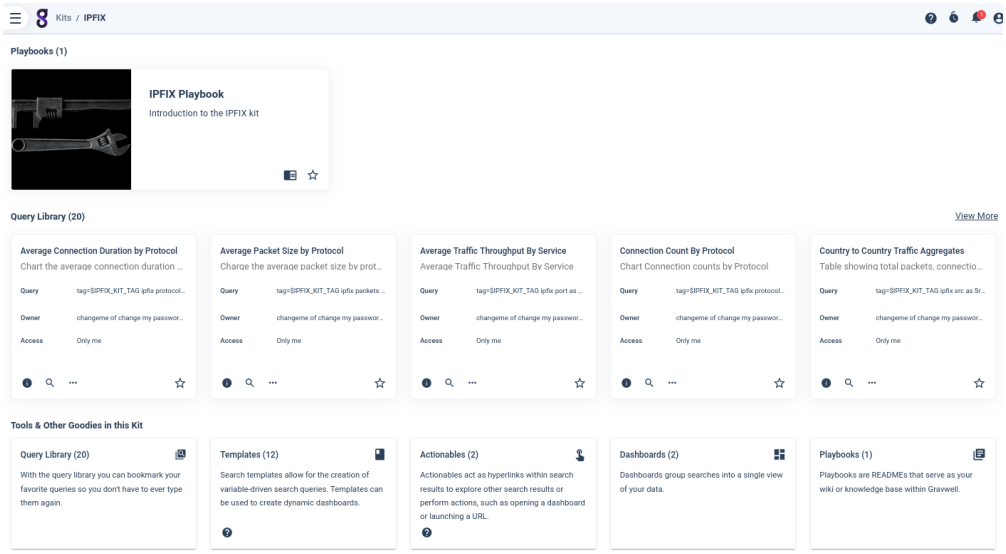


Figure 9.9: Exploring a kit

Note that opening the kit like this (or clicking its icon in the list of installed kits) puts you in the kit's *context*. You can tell you are inside a kit context because the text "Kits / IPFIX" is in the top bar of the UI. While you're in a kit context, the UI will only show items contained within the kit. To leave the kit context, either click the Gravwell logo at the top of the page, or click 'Exit kit' in the main menu.

You can always re-visit an installed kit by opening the kit menu and clicking the kit's icon in the list of installed kits.

9.3 Managing Installed Kits

9.3.1 Upgrading Kits

Once a kit has been installed, little administration is required. The sole point of manual intervention required is *upgrading* a kit when a new version comes out. Gravwell will periodically push updates to the official kit server. When one of your installed kits has an update available, an "Upgrade" button will appear on that kit's tile, as shown in Figure 9.10

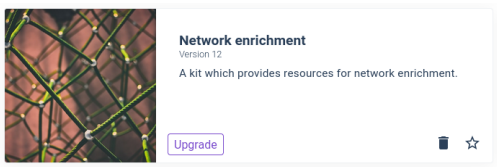


Figure 9.10: The kit upgrade notification

Clicking the button will launch an upgrade wizard similar to the installation wizard. The most important difference is the Backup option. If you have modified any of the items which were included in the kit, the wizard will notify you and provide options for copying or downloading the modified items. If there are no

items which have been modified, the Backup step will not be shown. The rest of the wizard is identical to the installation wizard, although defaults such as group access should be already set for you.

Be warned that upgrading a kit to a new version involves the complete deletion of the previous version's contents. Do not click the "Deploy" button at the end of the wizard until you are prepared for this to happen!

9.3.2 Uninstalling Kits

To remove an installed kit, enter kit management mode by clicking the "Manage Kits" button in the upper-right corner of the main kits page. Then select the trash can icon on the desired kit. A dialog will pop up for confirmation, as shown in Figure 9.11. If you then click "Uninstall", the kit will be removed, *unless* you have manually changed any of the kit contents. If you have modified any of the kit items, you will see a second dialog warning you of this fact and allowing one last chance to abort the process, as seen in Figure 9.12

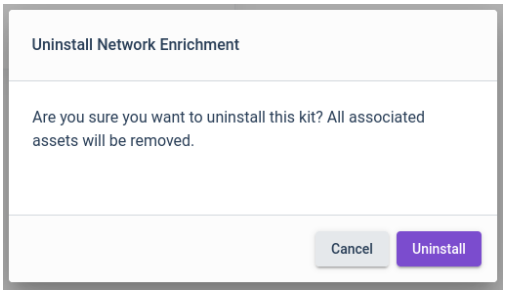


Figure 9.11: Kit deletion dialog

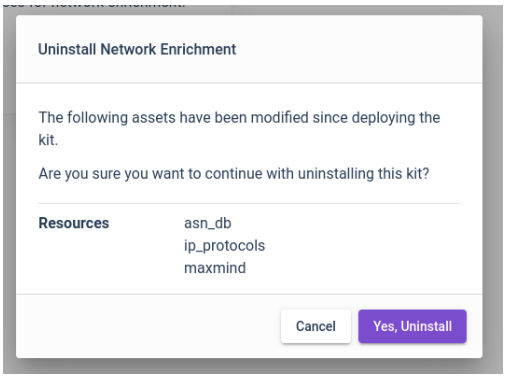


Figure 9.12: Kit deletion warning dialog for modified items

9.4 Hands-on Lab: Installing Kits

For this hands on lab we are going to install the Netflow kit, browse some of the kit contents, then uninstall the kit.

Start by cleaning your environment and starting up a new base Gravwell instance:

```
docker stop $(docker ps -q)
docker rm $(docker ps -q -a)
docker create --rm --net gravnet -p 8080:80 --name kits gravwell:base
docker start kits
```

Next, start the ingester container running the netflow ingester:

```
docker run --rm --net gravnet --name ingesters -it \
-e GRAVWELL_CLEARTEXT_TARGETS=kits:4023 \
gravwell:ingesters /opt/gravwell/bin/gravwell_netflow_capture
```

The netflow ingester is pre-configured to listen on port 2055 for incoming Netflow v5 records.

Now, we use another Docker container to generate Netflow records and send them to the ingester:

```
docker run -it --net gravnet --rm \
networkstatic/nflow-generator -t ingesters -p 2055
```

The netflow generator will run indefinitely, generating flow records, until killed.

Log into your Gravwell GUI (http://localhost:8080) and navigate to the Kits page. Click 'Manage kits', which should take you to the list of available kits. Find and click the deploy button on the Netflow v5 kit, then walk through the installation wizard.

When installation is done, click the 'Explore the kit' button, then open the playbook and run some of the sample queries on the Netflow data we're ingesting. Explore the rest of the kit, then try to answer these questions:

1. How many dashboards are included in the kit?

2. Click the Gravwell icon at the top of the page to leave the kit context. How do you now get back into the kit contents list?

3. Open the Netflow V5 Traffic Overview dashboard, scroll down to the 'Rare Ports' tile, and click one of the IP addresses. If you wanted to add to the actionables, how would you find the actionable definitions?

When you're done, return to the 'Manage kits' page and uninstall the kit. Verify that none of the Netflow dashboards or other objects still exist. Note that there is still a kit installed–which kit is it, and why was it installed?

To clean up after the experiment, simply run:

```
docker kill $(docker ps -a -q)
```

9.5 Building Kits

Although Gravwell distributes pre-built official kits, any user can build a kit themselves. This is a convenient way to share objects built on one Gravwell instance with another instance. Note that kits built like this are not signed by Gravwell and therefore can only be installed by administrators.

You can build a kit by clicking the 'Build' button on the Manage Kits page. This launches the kit-building wizard. On the first page, seen in Figure 9.13, you set general options. The name should be a user-friendly short name like "Network Enrichment", "Zeek Kit", etc. The kit ID should be a namespaced and unique ID for your kit; although the field is free-form, we recommend using domain namespaces, e.g. "io.gravwell.example". The Version field sets a version for the kit, which is useful for upgrades if you decide to set up your own kit server. The optional Gravwell minimum and maximum versions allow you to restrict the kit's compatiblity to particular versions of Gravwell. Finally, the kit icon field lets you optionally set a small image which may be used by Gravwell to help identify the kit to users.

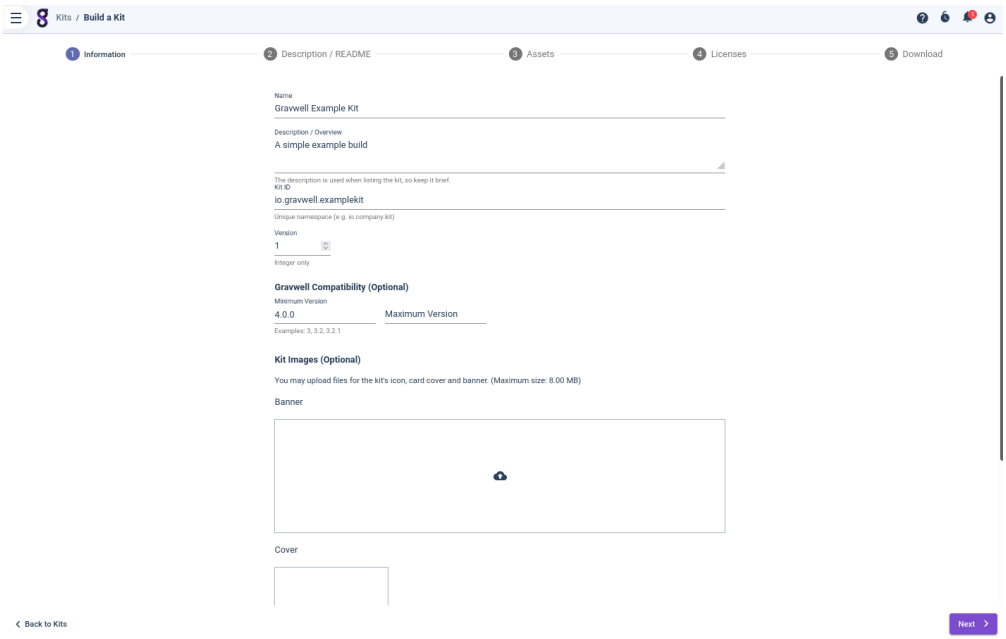


Figure 9.13: Build wizard, general options

The next page (Figure 9.14) is a markdown editor in which you can write a detailed description of the kit. You can go as simple or as complex as you like, but try to add some detail.

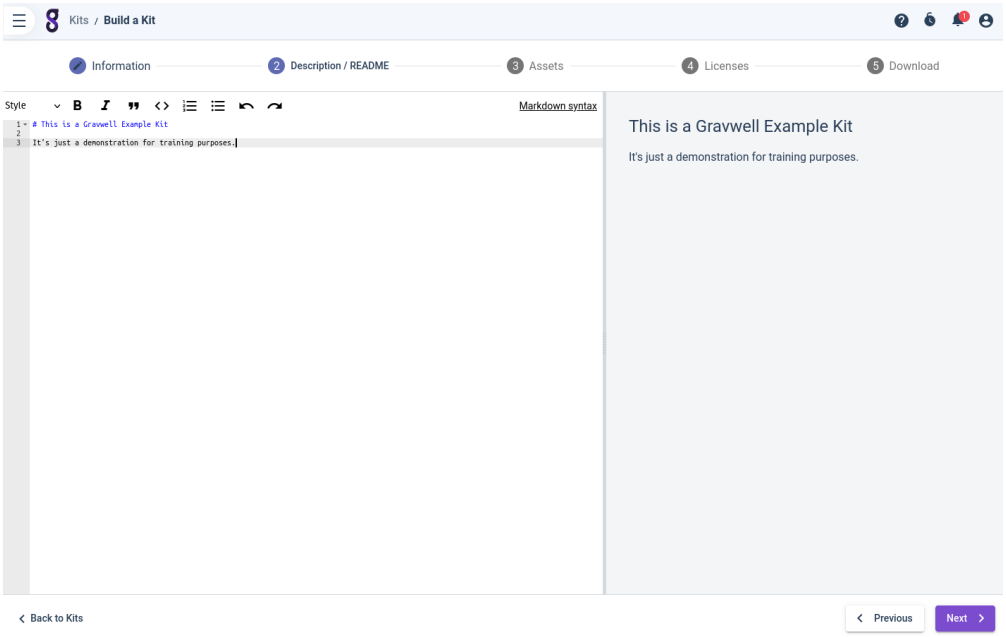


Figure 9.14: Build wizard, description page

The third page, seen in Figure 9.15, is where you actually select what goes in the kit. You can select an asset category on the left (dashboards, resources, macros, playbooks, etc.), then click to select or deselect items within that category on the right. In the screenshot, we are viewing the Query Library entries and have selected the first 5. We have also previously selected one dashboard.

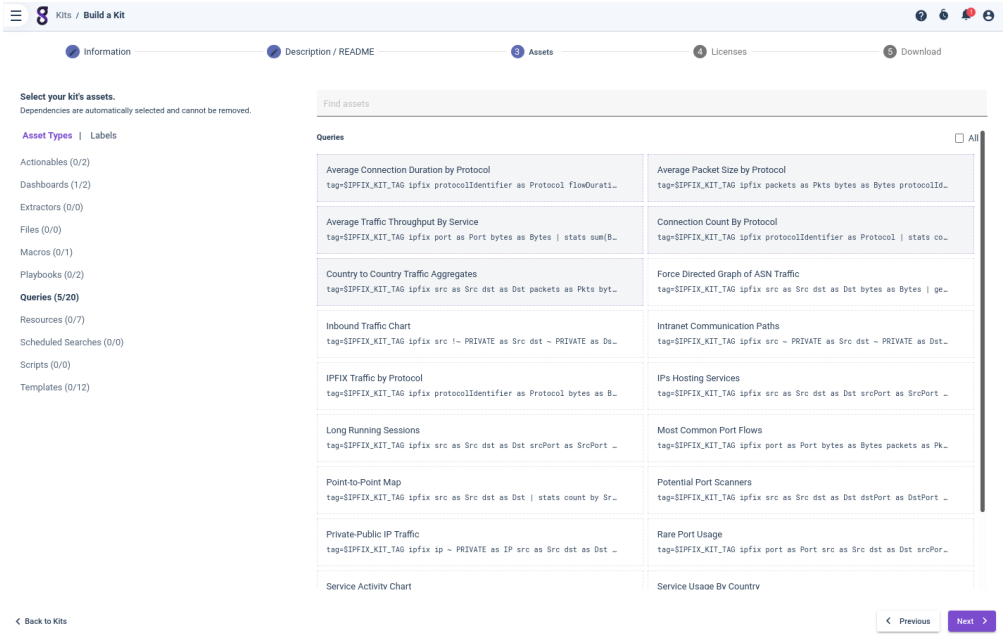


Figure 9.15: Build wizard, asset selection

After selecting assets comes the license page (Figure 9.16), where you can add licenses if needed–for instance, if you were bundling a third-party resource which requires a license.

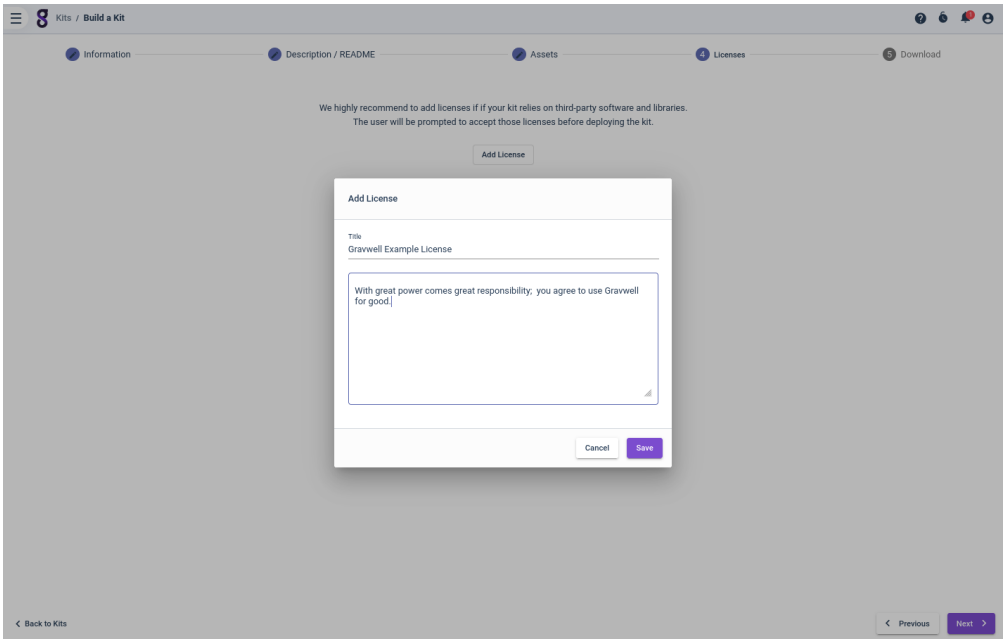


Figure 9.16: Build wizard, license page

After selecting assets, the only step remaining is to download the completed kit. Figure 9.17 shows this page. Note that if you don't click "Download", the in-progress kit will not be saved anywhere! You can go back and make changes to any of the previous steps before downloading by clicking the step at the top of the page.

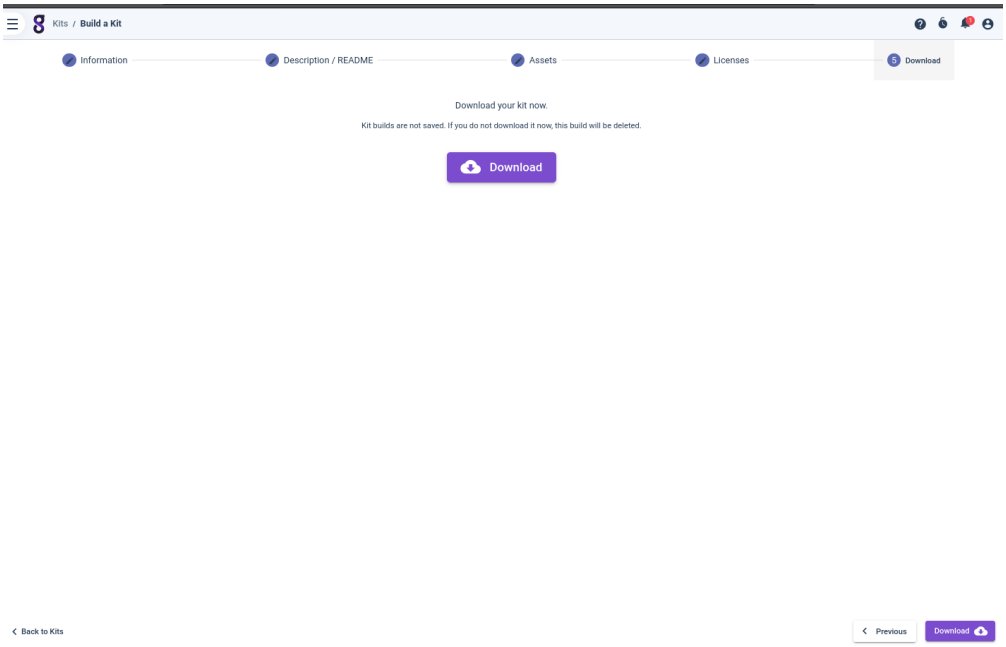


Figure 9.17: Build wizard download

Chapter 10

User and Group Management

Gravwell implements users and groups in a manner very similar to Unix. Each user has a username, a real name, and a numeric UID. Each group has a name, a numeric group ID, and a list of member UIDs. As in Unix, users can belong to multiple groups. The biggest difference from Unix is that a given item (such as a resource) can often be made accessible to members of *multiple* groups, where Unix only allows one.

Many user actions generate entries in the webserver's logs. Provided the webserver's Log-Level parameter has been set to `INFO`, the file `/opt/gravwell/log/web/info.log` will contain very verbose logs of user logins, user searches, and more.

10.1 Managing Users

The Gravwell GUI includes an admin-only page for managing users, located in the menu under the Administrator section. From this page, an administrator can create new users, modify existing users, or delete users. Figure 10.1 shows the User page on a freshly-installed Gravwell system; the only extant user is the default "admin".

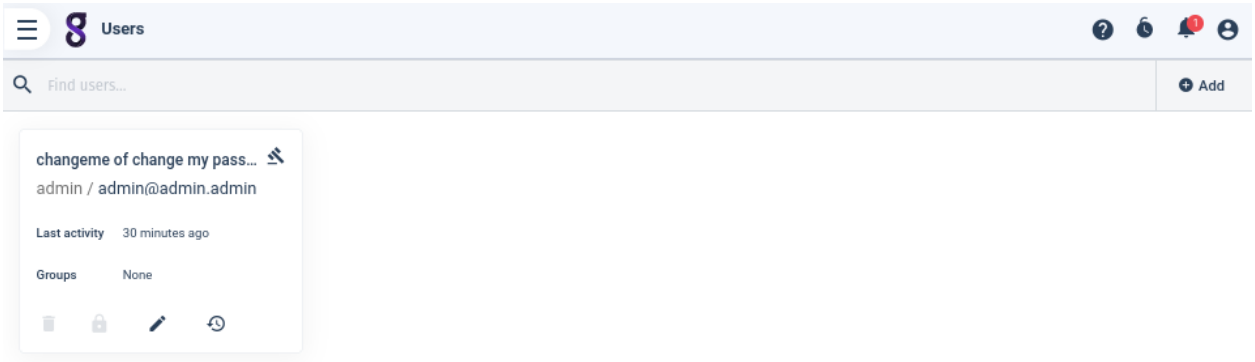


Figure 10.1: A Freshly-Installed System's User List

Clicking the 'Add' button in the upper right brings up a dialog to create a new user, as shown in Figure 10.2. Note the 'Administrator' checkbox at the bottom of the dialog. If this box is checked, the user will receive administrator-level privileges. Take care when selecting this option!

Each user information tile has four icons across the bottom, as shown in Figure 10.3. Selecting the trashcan icon will delete the user (prompting for confirmation). The padlock icon will lock the user account, logging

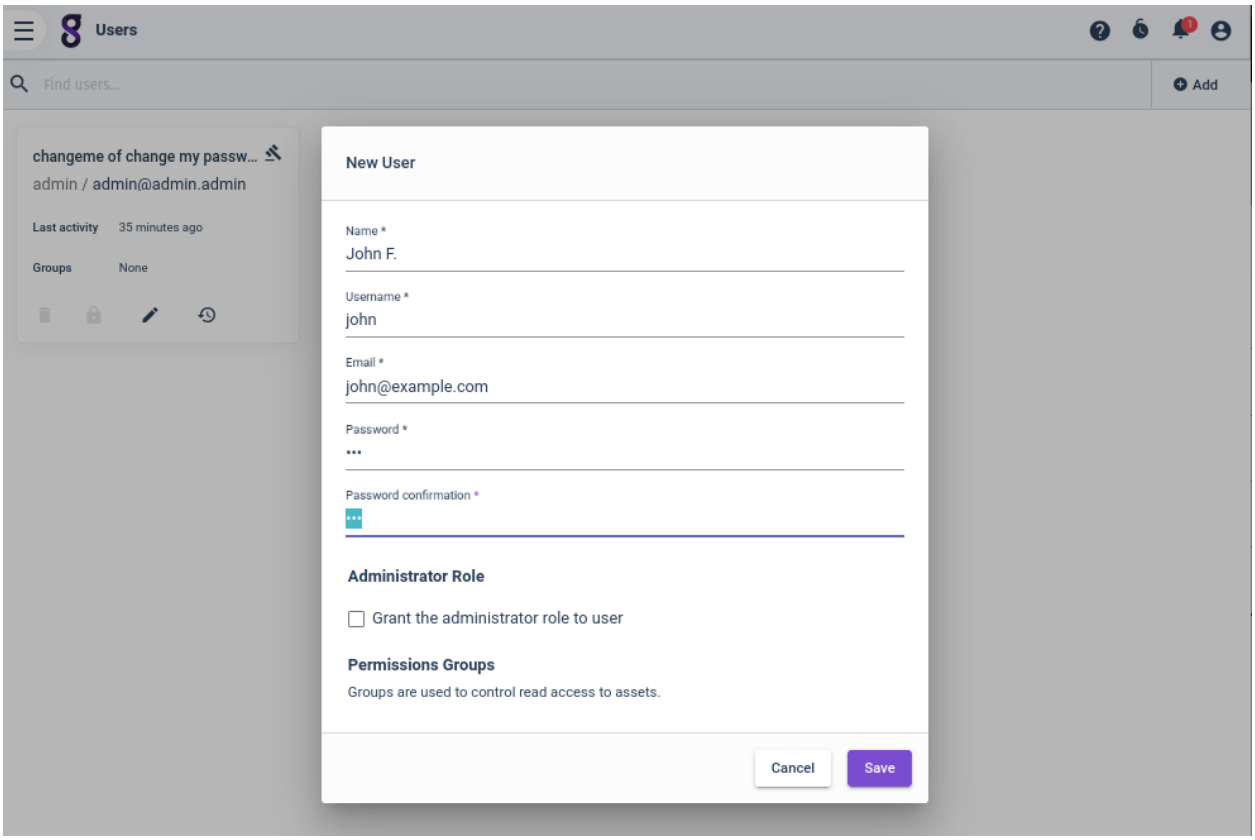


Figure 10.2: New User Dialog

out any sessions they may currently have and preventing them from logging in again until the account is unlocked; this is a good way to deal with misbehaving users. Both of these icons are disabled for the 'admin' user.

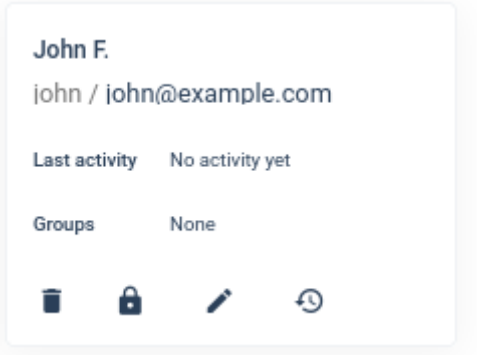


Figure 10.3: The User Information Tile

The pencil icon brings up a dialog to edit the existing user, as shown in Figure 10.4. This allows you to reset the user's password, change their personal information, or even make them an administrator if needed.

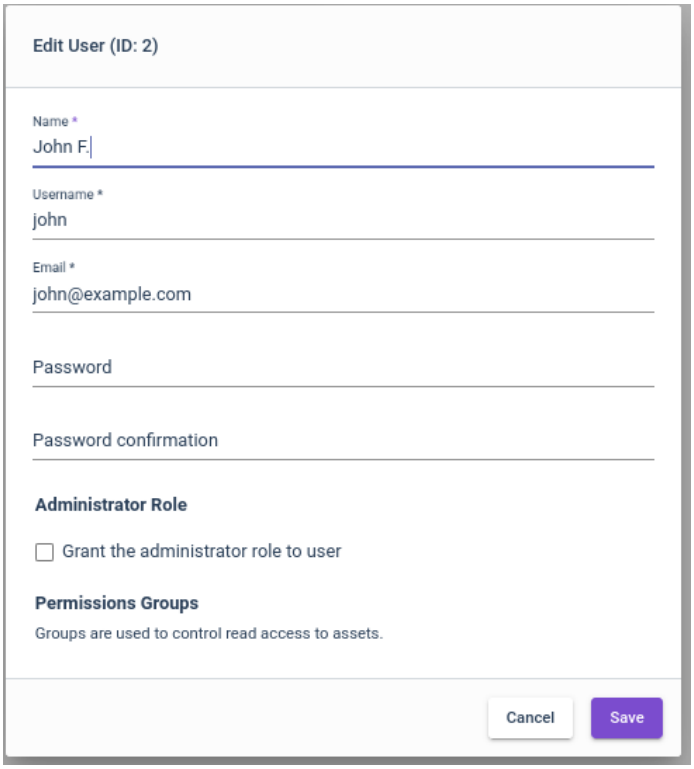


Figure 10.4: Editing User Information

Finally, the clock icon brings up the user's search history, as shown in Figure 10.5. This can be useful when attempting to help a user figure out why their searches aren't working, or to keep an eye on what users are trying to extract from your data.

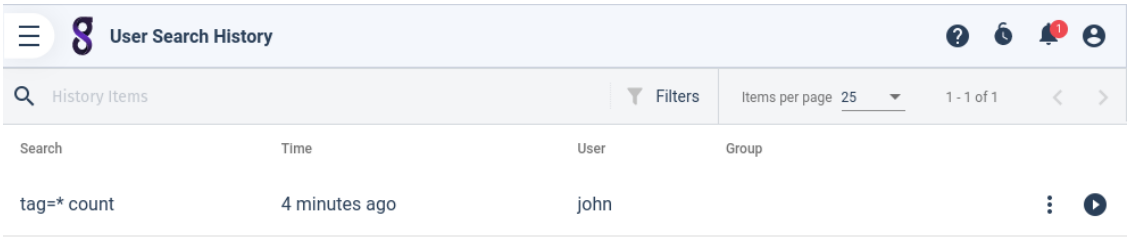


Figure 10.5: User Search History

10.2 Managing Groups

Group management is very similar to user management. The Groups page in the Administration section of the menu will initially show no groups, as in Figure 10.6

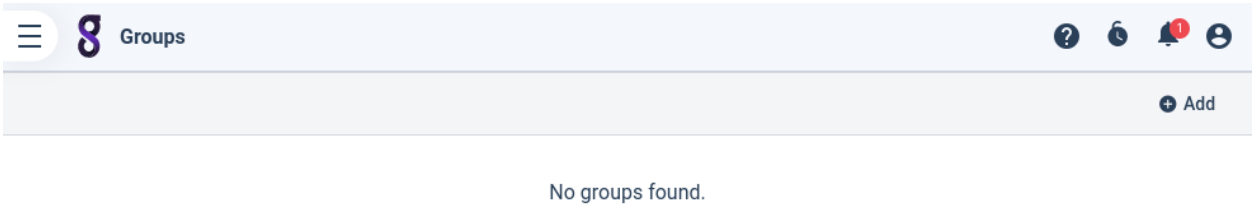


Figure 10.6: A Freshly-Installed System's Group List

A group can be added by clicking the 'Add' button and filling out the form, optionally selecting any users which should be members of the new group, as shown in Figure 10.7

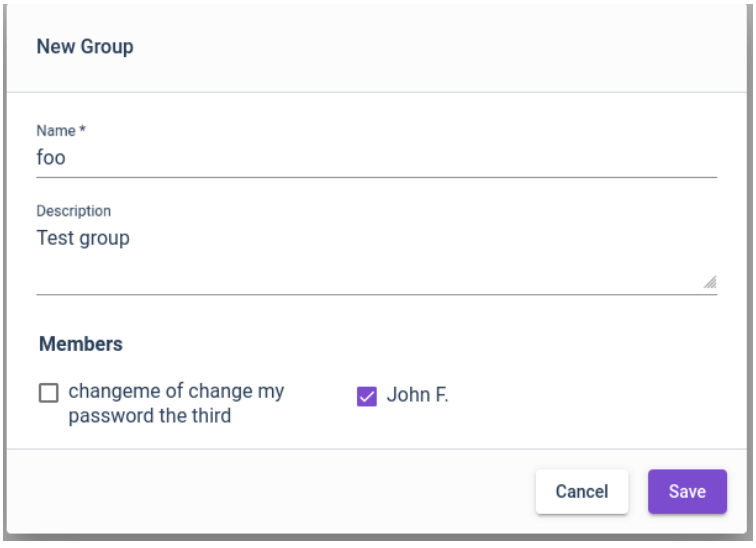


Figure 10.7: New Group Dialog

Once created, a group's tile has three action icons as seen in Figure 10.8. The trashcan icon deletes the group. The pencil icon allows the group to be renamed or for members to be added/deleted. The clock icon shows

the history of searches which have been run which were accessible to this group.

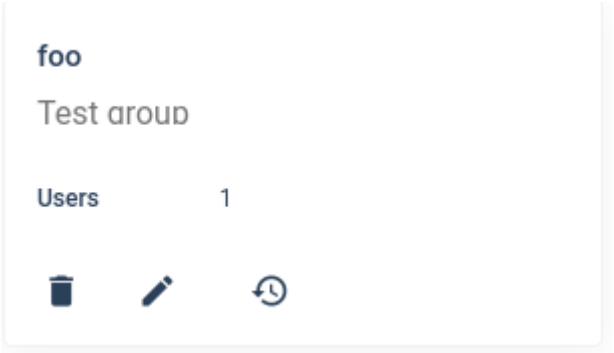


Figure 10.8: The Group Information Tile

10.3 Hands-on Lab: Managing Users and Groups

First, launch a Gravwell webserver+indexer container:

```
docker run --rm --net gravnet -p 8080:80 -d --name gravwell gravwell:base
```

Log into the web GUI (http://localhost:8080) and try the following tasks using the GUI:

1. Create a new user for yourself.
2. Create a new group and add your user to the group.
3. Log in as your user in a separate incognito window. From the admin account, lock your user account. What happened?

To clean up after the experiment, run:

```
docker kill $(docker ps -a -q)
```

Chapter 11

Migration

When you first stand up Gravwell, one of the first tasks is typically getting data from an old system or log archive into Gravwell. Migration may involve pulling historical data out of an existing system like Splunk, scraping a large database, or just ingesting 5 years of old syslog from flat files. Gravwell provides a series of "normal" ingesters designed to stream data in from a variety of sources in near real-time, but for one-time migration of existing data, there are specialized tools which may be a better choice.

This section will explore the available tools to migrate existing data and give some tips on how to efficiently migrate potentially hundreds of TB of existing logs into a new Gravwell instance. We will examine a few scenarios where using migration tools will provide much better migration performance and storage efficiency as opposed to just tossing something at a typical ingester. Most of our migration tools are open source, so we can also provide links to source code and deep dive examinations of functionality. Most data migrations are a one time occurrence, which means a one-off tool that is specialized for the task is usually the right answer.

11.1 The Interactive Migration Tool

Gravwell provides an interactive tool for migrating text files and Splunk data into Gravwell. This section describes how to install, configure, and use it.

11.1.1 Installation

The migration tool is included in the `gravwell-tools` package for Debian and Redhat, and is also available as a standalone shell installer. Once installed, the program is located at `/usr/local/sbin/gravwell_migrate`.

11.1.2 Basic Configuration

The migration tool stores its configuration in two different places:

- A top-level config file, by default located at `/opt/gravwell/etc/migrate.conf`.
- A config directory, by default `/opt/gravwell/etc/migrate.conf.d`, which contains automatically-generated config snippets stored by the program.

These files are owned by the `gravwell` user. Both of these locations may be specified manually using the `-config-file` parameter to point at your main config file, and `-config-overlays` to point at a directory you wish to use for configuration snippets, which is useful if you are unable to execute the migrate command as the Gravwell user.

In general, you should only need to modify `migrate.conf`. The following is a simple configuration which migrates data from both Splunk and from files on disk:

```
[Global]
Ingester-UUID="0796e339-bd04-4dbf-be8d-f92fa5b08792"
Ingest-Secret = IngestSecrets
Connection-Timeout = 0
Insecure-Skip-TLS-Verify=false
Cleartext-Backend-Target=192.168.1.50:4023
State-Store-Location=/opt/gravwell/etc/migrate.state
Log-Level=INFO
Log-File=/opt/gravwell/log/migrate.log

[Splunk "splunk1"]
Token=`eyJraWQiOj[...]nlHnn4Oivew`
Server=splunk.example.org

[Files "auth"]
Base-Directory="/var/log"
File-Filter="auth.log,auth.log.[0-9]"
Tag-Name=auth
Recursive=true
Ignore-Line-Prefix="#"
Ignore-Line-Prefix="//"
Timezone-Override="UTC" #force the timezone
```

It specifies:

- Data should be ingested to the Gravwell indexer at `192.168.1.50:4023`, using 'IngestSecrets' as the token to authenticate with Gravwell.
- There is a Splunk server at `splunk.example.org` which can be accessed using the given token (the token has been shortened for this document).
- It should pull `auth.log`, `auth.log.1`, `auth.log.2` and so on from `/var/log` and ingest each line as an entry, using the Gravwell tag "auth".

The sections on migrating files and migrating Splunk data have detailed descriptions of the configuring each migration type; see below.

11.1.3 Launching the Tool

To start the tool, just run the provided binary *as the gravwell user*:

```
/usr/local/sbin/gravwell_migrate
```

If you cannot execute the program as the `gravwell` user, you'll need to create your own configuration file and config directory, then specify them via command-line options (see section 11.1.2 for more details):

```
/usr/local/sbin/gravwell_migrate -config-file ~/migrate.conf -config-overlays ~/migrate.conf.d/
```

If the configuration is valid, you should see the main menu of the migrate tool (see screenshot below). The UI displays several panes of information: the main menu where you select actions, the "Jobs" pane where running migrations are tracked, the "Logs" pane where debugging information is displayed, and the "Help" pane which shows some basic key combinations (see Figure 11.1.

11.1.4 Migrating Files

Importing files from the disk is quite simple. First, set up the configuration file to point at files on the disk you're interested in ingesting. Then, from the main menu, select the "Files" option. You should see a list of all the `Files` config blocks you defined. For instance, given the following configuration block, you should see a menu which resembles the screenshot in Figure 11.2.

Figure 11.1: The migration tool UI

```
[Files "auth"]
Base-Directory="/tmp/log"
File-Filter="auth.log,auth.log.[0-9]"
Tag-Name=auth2
Ignore-Timestamps=true #do not ignore timestamps
Recursive=true
```

Selecting “auth” opens another menu from which the migration can be launched. To begin the migration, press the ‘s’ key or select the “Start” option. As seen in Figure 11.3, a job will appear in the Jobs pane showing the migration progress. In this case, there were 2 files ingested, for a total of 1444 entries and 141537 bytes of data.

Note that there are actually two jobs shown in the screenshot. After the first migration job completed, “Start” was selected again. However, the migration tool tracks how much of each file it has ingested, so it will not duplicate data; the second job simply noted that there was no new data to ingest and exited.

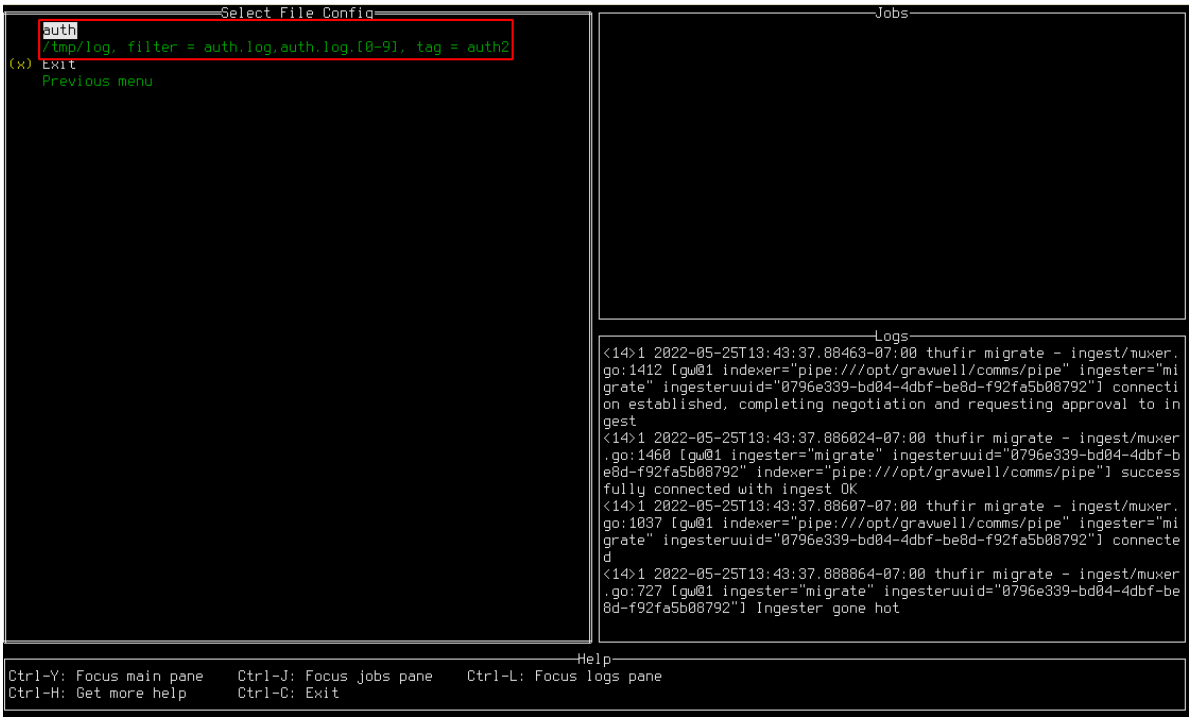


Figure 11.2: File migration menu

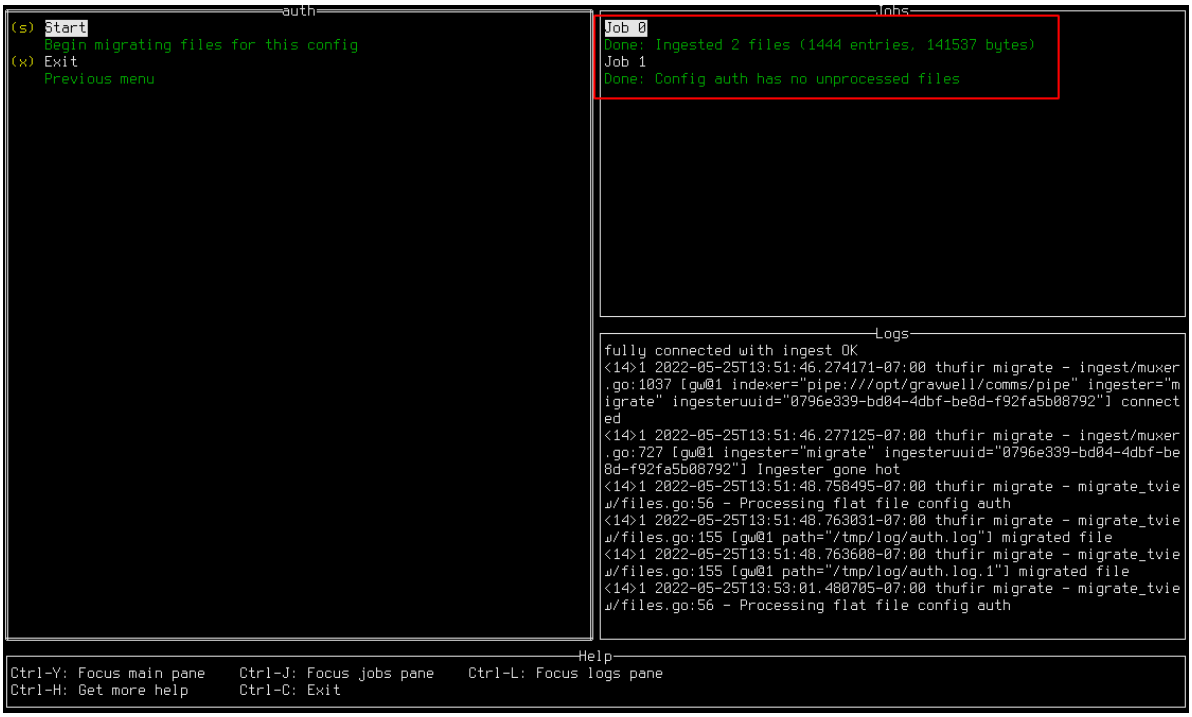


Figure 11.3: File migration jobs

File Configuration Details

The migrate tool can have multiple `Files` configuration blocks which allows the migrate tool to consume from many different directories, applying tags, custom timestamps, and even custom preprocessors to different batches of files. If you are performing a very large migration from many different file sources, you can set up a big config file that points at all your sources, fire up the migration jobs, and head home for the weekend. The 'migrate' tool will walk each directory and consume each file according to the given configuration.

A `Files` configuration target is defined using the `Files` specifier with a unique name. Here is an example looking for files in `/tmp/logs`:

```
[Files "auth"]
Base-Directory="/tmp/logs/"
File-Filter="auth.log,auth.log.[0-9]" #we are looking for all authorization log files
Tag-Name=auth
```

The `Files` configuration element can contain the parameters in Table 11.1.

Here is an example config snippet which shows the full range of config options for a directory, including the use of a preprocessor:

```
[Files "testing"]
Base-Directory="/tmp/testlogs/"
File-Filter="app.log,app.log.[0-9],host.log*"
Tag-Name=apps
Assume-Local-Timezone=true #Default for assume localtime is false
Ignore-Timestamps=false
Recursive=true
Ignore-Line-Prefix="#"
Ignore-Line-Prefix="//"
Preprocessor="loginapps"

[Preprocessor "loginapp"]
Type=regexextract
Regex="\\S+ (?P<host>\\S+) \\d+ \\S+ \\S+ (?P<data>\\{.+\\})$"
Template="{\"host\": \"${host}\", \"data\": ${data}}"
```

Parameter	Required	Description	Example
Base-Directory	yes	A full path to the directory containing flat files to be ingested.	`Base-Directory=/var/log/auth`
File-Filter	yes	A comma separated list of file glob patterns that specify which files to consume.	`File-Filter="*.log,file.txt"`
Tag-Name	yes	The tag name that the migrate tool should ingest all files into. This must be a valid tag name.	`Tag-Name=auth`
Ignore-Timestamps		A Boolean value indicating if the migrate tool should not attempt to resolve timestamps, but should instead use the timestamp of now. The default value is false.	`Ignore-Timestamps=true`
Assume-Local-Timezone		A Boolean indicating that if the resolved timestamps do not have a timezone, use the local timezone. The default is false, meaning use UTC.	`Assume-Local-Timezone=true`
Timezone-Override		A string indicating that if the resolved timestamps do not have a timezone, use the given timezone. The timezone must be specified in IANA format.	`Timezone-Override="America/New_York"`
Recursive		A Boolean indicating that the tool should recurse into any sub-directories found in the `Base-Directory` and attempt to match and consume files. Default is false.	`Recursive=true`
Ignore-Line-Prefix		A string which indicates that a line should be ignored. This can be specified multiple times and is useful for dealing with headers on files.	`Ignore-Line-Prefix="#"`
Preprocessor		Specify preprocessors to be applied to entries as they are consumed from flat files. More than one preprocessor can be specified and they are executed in order.	`Preprocessor="logins"`

Table 11.1: Files configuration options

11.1.5 Migrating Splunk Data

The migrate tool can copy data out of a Splunk server and into Gravwell. In Splunk, entries are organized by index and source type; in the migration process, the tool pulls entries out of a given index & sourcetype and ingests them into a Gravwell tag.

Configuring Splunk Tokens

In order to fetch data from a Splunk server, you must generate an authentication token which the migration tool can use to communicate with Splunk. Tokens may be generated in the Splunk UI under Settings > Tokens (Figure 11.4).

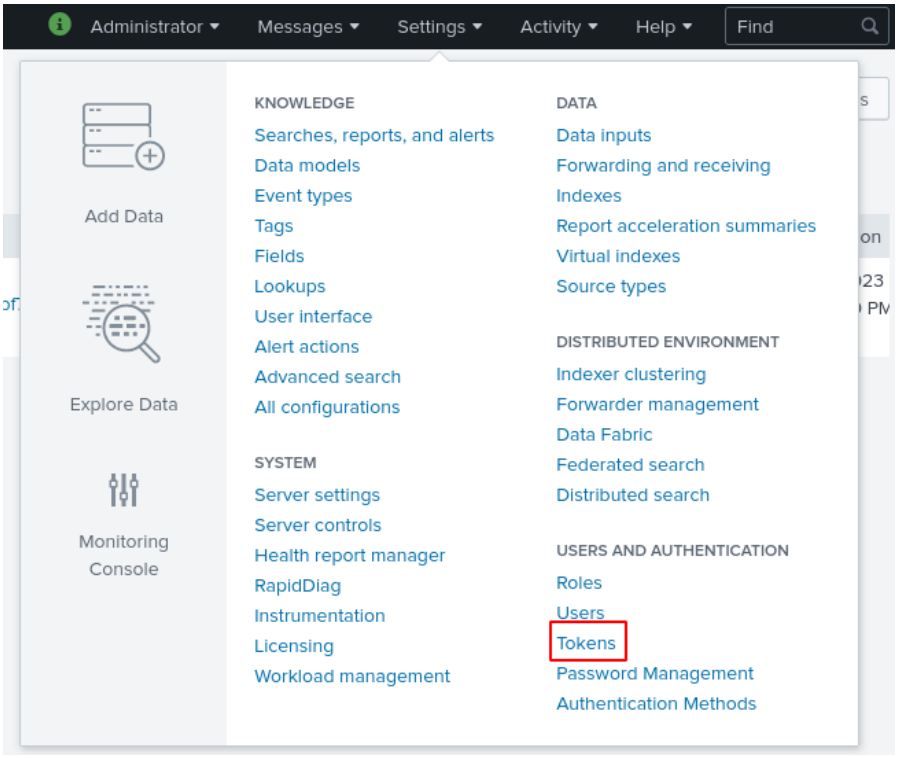


Figure 11.4: Splunk tokens menu

On the Tokens page, click the 'New Token' button, then fill in the "Audience" field with something like "Gravwell migration", select a token expiration time if desired (+60d is a good choice), and click 'Create'. The UI will then display a token in the "Token" field as seen in Figure 11.5; copy this and save it somewhere, because it cannot be retrieved later!

This token string should be inserted into the `Token` field of a Splunk configuration block in the main config file. The `Server` field should correspond to whatever IP address or hostname you use to access your Splunk server.

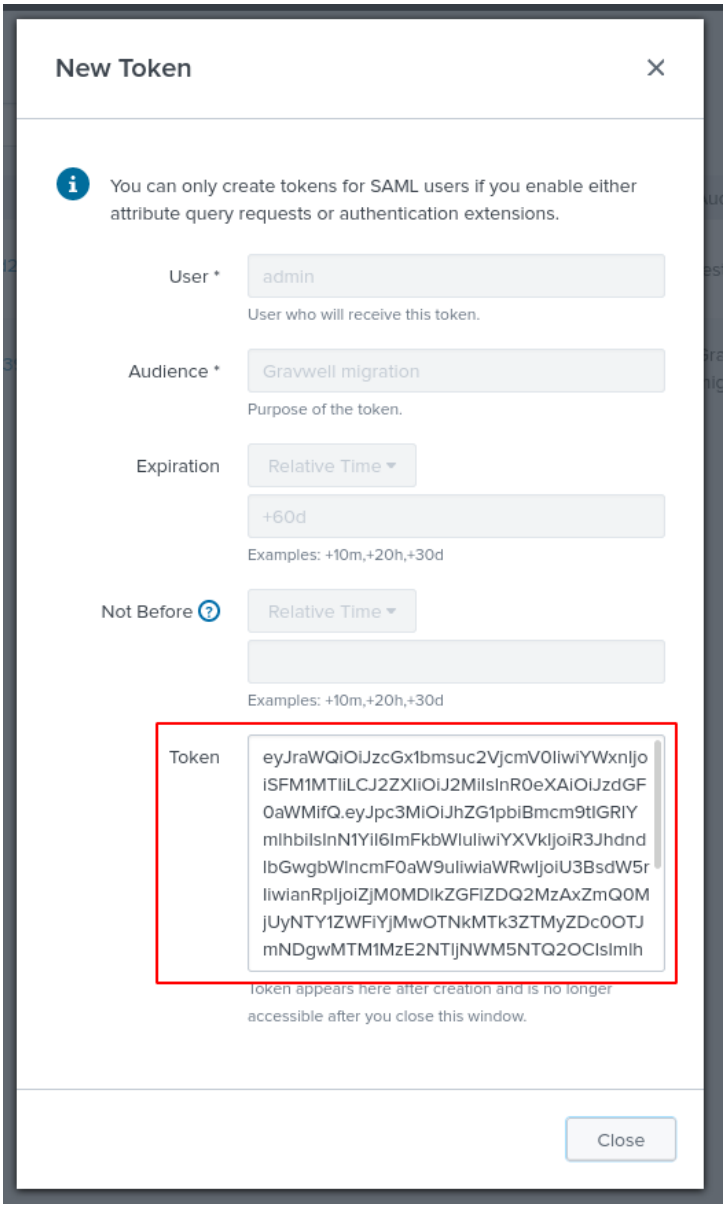


Figure 11.5: Newly-generated Splunk token.

Parameter	Required	Description	Example
Server	yes	The hostname or IP address of the Splunk server. Port 8089 must be accessible.	`Server=splunk.example.org`
Token	yes	A valid Splunk auth token.	`Token=eyJraWQiOj[...]nlHnn4Oivew`
Ingest-From-Unix-Time		A Unix timestamp which specifies the default "start" time to use when copying entries from the Splunk server. This may be overridden on a per-sourcetype basis.	`Ingest-From-Unix-Time=1625100000`
Index-Sourcetype-To-Tag		A mapping of a Splunk index and sourcetype pair to a Gravwell tag. Can be set interactively from within the migrate tool.	`Index-Sourcetype-To-Tag=main,json:impor` (maps the index "main" and sourcetype "json" to the Gravwell tag "importedjson")
Preprocessor		Specify preprocessors to be applied to entries as they are consumed from Splunk. More than one preprocessor can be specified and they are executed in order.	`Preprocessor="logins"`

Table 11.2: Splunk config options

Configuring Splunk Migrations

The migrate tool must know a few basic things about the Splunk server in order to connect:

- The server's IP or hostname.
- A valid auth token for the Splunk server.

These properties are defined in a `Splunk` configuration block within the config file. The simplest version might look like this, defining a server named "splunk1":

```
[Splunk "splunk1"]
Token=`eyJraWQiOj[...]nlHnn4Oivew`
Server=splunk.example.org
```

Multiple `Splunk` blocks may be defined, although each must have a unique name. The parameters shown in Table 11.2 may be used in a `Splunk` config block.

Note: The migrate tool will automatically create a file named `splunk1.conf` in `/opt/gravwell/etc/migrate.conf.d` to store your index+sourcetype→tag mappings (see below). Settings in this file will be merged with the settings in the main configuration file automatically.

Importing Splunk Data

To import data from Splunk, make sure you have configured at least one `Splunk` block in the config file (see configuration section above), then select "Splunk" from the main menu. This will open a new menu (Figure 11.6) in which you can select which Splunk server to migrate from.

Once a server has been selected, you will see the server's menu (Figure 11.7)

Figure 11.6: Splunk server selection.

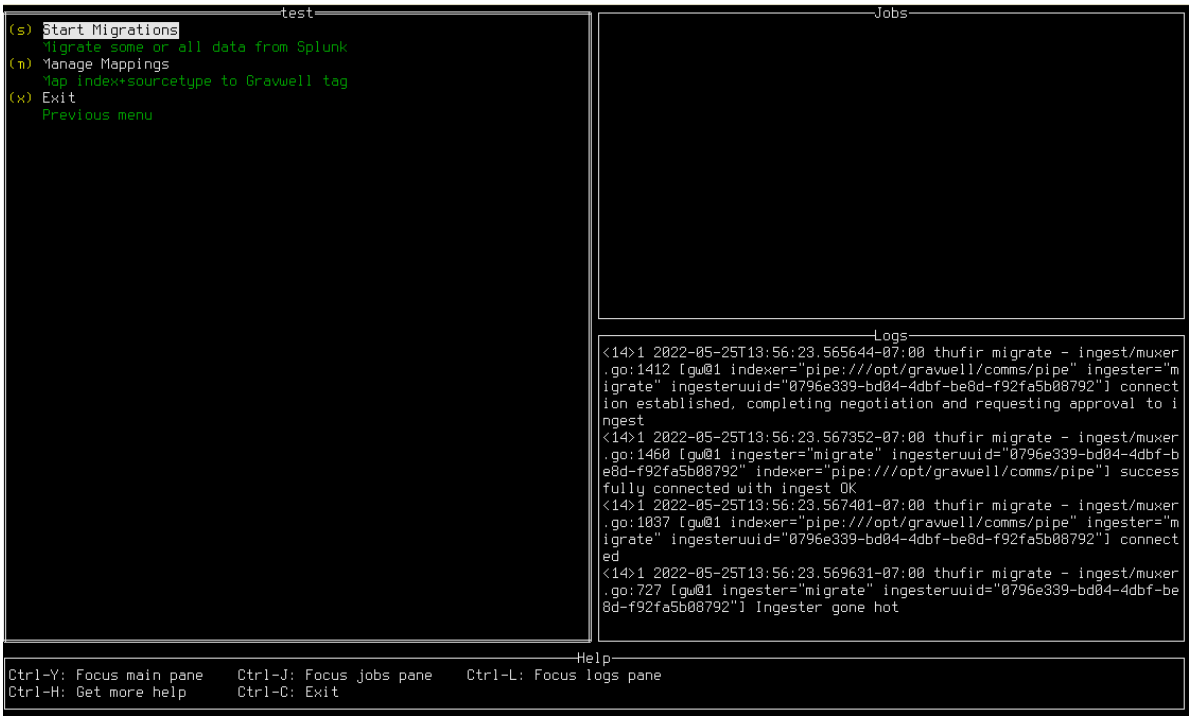


Figure 11.7: Splunk server menu.

Mapping index+sourcetype to tag

You must now define how Splunk's data organization should be mapped to Gravwell. In Splunk, data is organized into indexes and sourcetypes. In Gravwell, data simply receives a tag. To define these mappings, select "Manage Mappings"; this will open the mapping screen shown in Figure 11.8

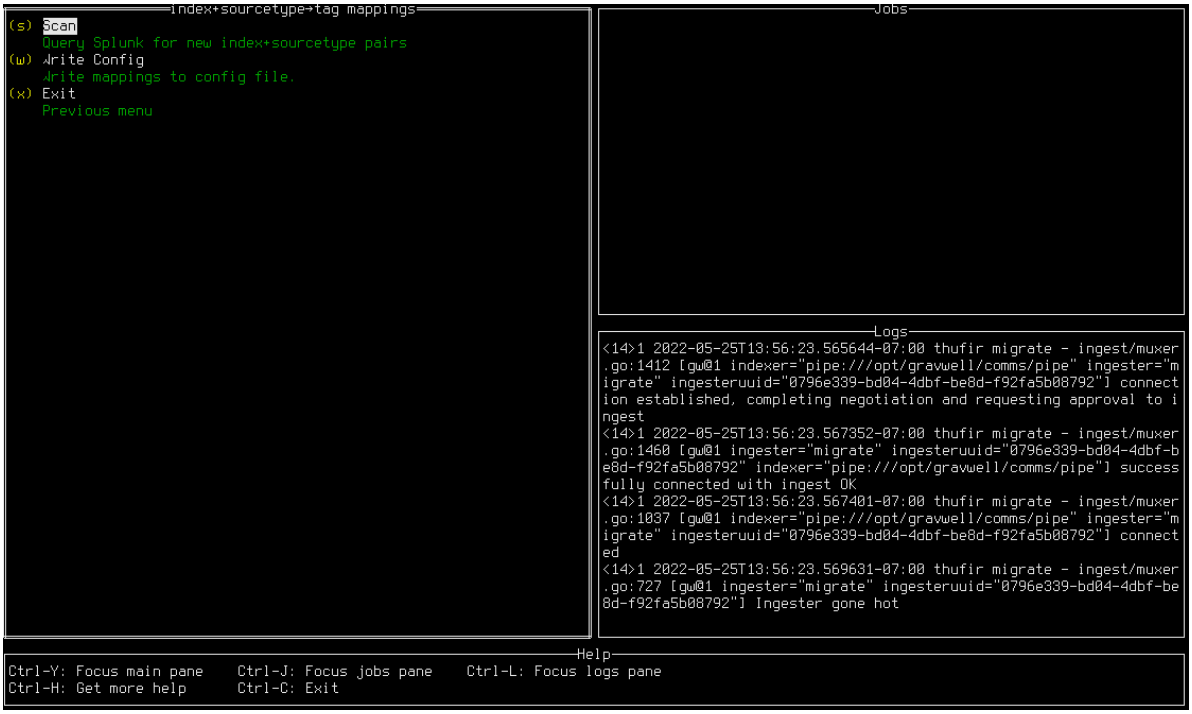


Figure 11.8: Mapping management menu.

Initially, the tool is not aware of which indexes and sourcetypes exist on the Splunk server. Select "Scan" to connect to the Splunk server and query this information; this may take a few seconds. Once the scan is complete, several index+sourcetype pairs should be visible, each with a blank tag, as seen in Figure 11.9.

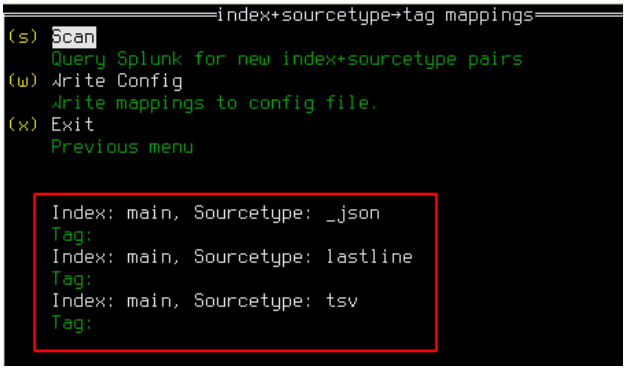


Figure 11.9: Sourcetype listing.

Select a pair which you wish to import and press enter. A form (Figure 11.10) will be displayed in which you may enter the Gravwell tag to be used; note that it will only allow you to type valid tag characters. You may also set a Unix timestamp to start the migration from, if you wish to exclude old data.

Note: If the Unix timestamp is set to 0, migration will begin from 1970, which can take a long time to complete even when no data is present. To speed things up, we strongly recommend either setting a timestamp

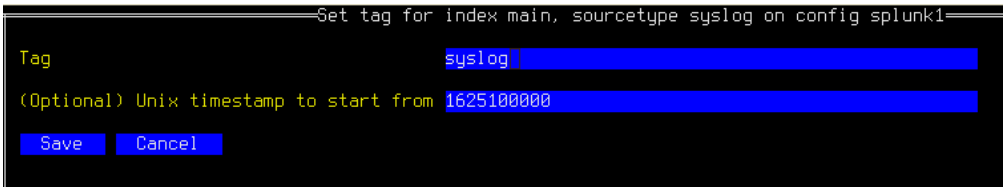


Figure 11.10: Tag definition form.

in this form, or setting the `Ingest-From-Unix-Time` parameter in the config file (see above).

After you have set the tag for the desired index + sourcetype pairs, you can select "Write Config" to write out a file in the `migrate.conf.d` directory which will store the mappings permanently.

Starting Migrations

Having defined mappings from Splunk index+sourcetype to Gravwell tag, you may now launch migration jobs. From the server menu, select "Start Migrations". A menu will appear showing the index+sourcetype → tag mappings you defined earlier. Selecting one of these mappings will start a migration job (Figure 11.11).

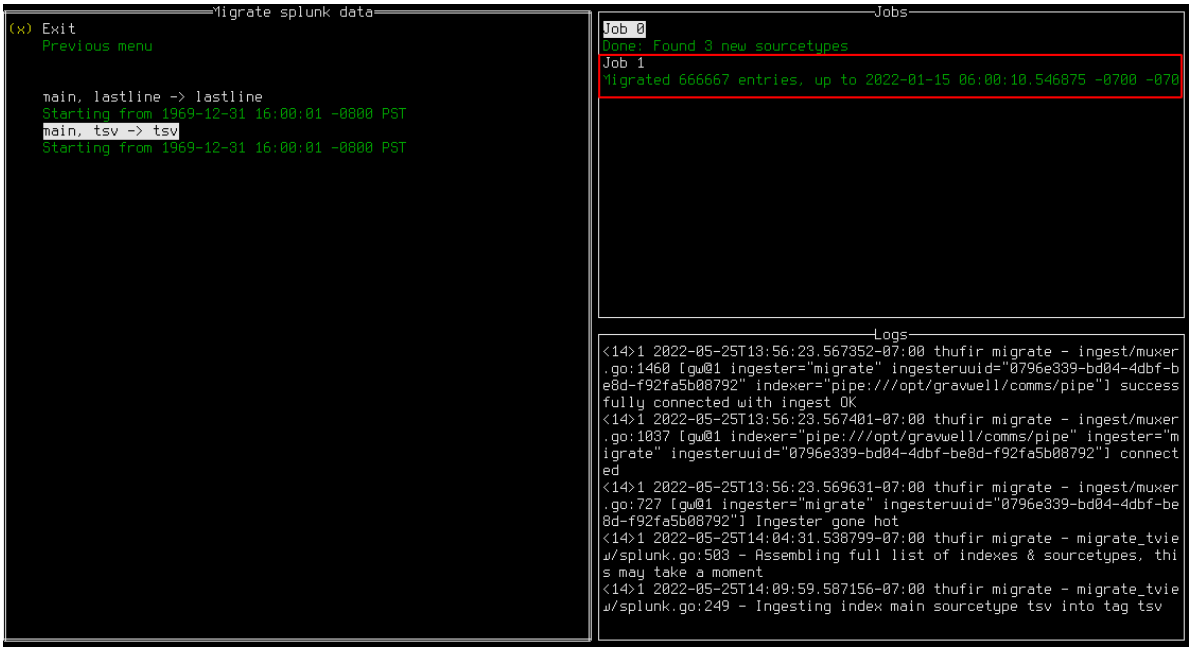


Figure 11.11: Migration jobs.

You can launch multiple migrations at once. Note that Splunk migrations may take a while; if you exit the migrate tool while a Splunk migration is running, the job will be halted as soon as possible and the most recent timestamp will be stored to resume later–we make every effort to avoid data duplication!

11.1.6 Quitting the Tool

You can exit the migration tool at any time by hitting Ctrl-C, or by backing all the way up to the top-level menu and pressing `q`. The tool will gracefully shut down all in-progress migrations and save its state. Note that if a particularly data-heavy Splunk migration job is in progress, it may take several seconds for the data to finish transferring; the UI will notify you that it is still waiting on some jobs and will quit when all are done.

11.2 Importing One File

The single file ingester is one of the most basic ingesters. It is designed to ingest a single line-delimited file to a specific tag. It can transparently decompress files and has some limited parsing ability. If you have a single large Apache access log or just need to script up some one off file ingestion, it can be the simplest option. The ingester is included in the 'gravwell-tools' package for Debian and Redhat, and is also available as a standalone shell installer. Once installed, the program is located at `/usr/local/sbin/gravwell_single_file_ingester`.

The ingester is a standalone ingester that is designed to operate using flags rather than a config file. This means that it lacks some of the additional functionality of other ingesters such as custom timestamp definitions and preprocessor support. Pass the `-h` flag to the command to get a listing of options:

```
-block-size int
   Optimized ingest using blocks, 0 disables
-clean-quotes
   clean quotes off lines
-clear-conns string
   comma-separated server:port list of cleartext targets
-i string
   Input file to process (specify - for stdin)
-ignore-prefix string
   Ignore lines that start with the prefix
-ignore-ts
   Ignore timestamp
-ingest-secret string
   Ingest key (default "IngestSecrets")
-pipe-conn string
   Path to pipe connection
-quotable-lines
   Allow lines to contain quoted newlines
-source-override string
   Override source with address, hash, or integer
-status
   Output ingest rate stats as we go
-tag-name string
   Tag name for ingested data (default "default")
-timeout int
   Connection timeout in seconds (default 1)
-timestamp-override string
   Timestamp override
-timezone-override string
   Timezone override e.g. America/Chicago
-tls-conns string
   comma-separated server:port list of TLS connections
-tls-private-key string
   Path to TLS private key
-tls-public-key string
   Path to TLS public key
-tls-remote-verify
   Validate remote TLS certificates (default true)
-utc
   Assume UTC time
-verbose
   Print every step
-version
```

```
    Print version and exit
```

Most of these flags are optional, but the following are required: At least one indexer must be specified, using `-clear-conns`, `tls-conns`, `pipe-conn`, or a combination thereof. The ingest secret should also be specified with `-ingest-secret`, and the desired tag should be specified with `-tag`. Finally, the file to ingest must be specified using the `-i` flag.

The following command ingests the contents of `/tmp/my-logs.txt`, one entry per line. It ignores (does not ingest) any lines starting with the `#` character. It will attempt to extract timestamps from the log entries (the default) but if no timezone is specified in the log entry, it will assume America/Chicago. The entries will be ingested to the indexer at `10.0.0.50`.

```
/usr/local/sbin/gravwell_single_file_ingester -clear-conns 10.0.0.50 -ingest-secret xyzzy123 -timezone-
```

11.3 Importing PCAP Files

If you have existing PCAP files (from Wireshark or tcpdump or some other packet capture tool), you can ingest them using the PCAP file ingester. The ingester is included in the 'gravwell-tools' package for Debian and Redhat, and is also available as a standalone shell installer. After installation, the program is found in `/usr/local/sbin/gravwell_pcap_file_ingester`.

The ingester is a standalone ingester that is designed to operate using flags rather than a config file. This means that it lacks some of the additional functionality of other ingesters such as custom timestamp definitions and preprocessor support. Pass the `-h` flag to the command to get a listing of options:

```
-clear-conns string
   comma-separated server:port list of cleartext targets
-ingest-secret string
   Ingest key (default "IngestSecrets")
-no-ingest
   Do not ingest the packets, just read the pcap file
-pcap-file string
   Path to the pcap file
-pipe-conn string
   Path to pipe connection
-source-override string
   Override source with address, hash, or integer
-tag-name string
   Tag name for ingested data (default "default")
-timeout int
   Connection timeout in seconds (default 1)
-tls-conns string
   comma-separated server:port list of TLS connections
-tls-private-key string
   Path to TLS private key
-tls-public-key string
   Path to TLS public key
-tls-remote-verify
   Validate remote TLS certificates (default true)
-ts-override
   Override the timestamps and start them at now
-version
   Print the version information and exit
```

Most of these flags are optional, but the following are required: At least one indexer must be specified, using `-clear-conns`, `tls-conns`, `pipe-conn`, or a combination thereof. The ingest secret should also be specified

with `-ingest-secret`, and the desired tag should be specified with `-tag`. Finally, the ingest file must be specified using the `-pcap-file` flag.

The following ingests the packets in `/tmp/netflow-capture.pcap` to an indexer on the local machine. Each packet is ingested as a single entry.

```
/usr/local/sbin/gravwell_pcap_file_ingester -pipe-conns /opt/gravwell/comms/pipe \
-ingest-secret MyIngestSecret -pcap-file /tmp/netflow-capture.pcap
```

Chapter 12

Command Line Interface

In addition to the GUI we've shown so far, Gravwell provides a command-line client. This client can be useful for certain repetitive tasks such as testing scripts (see the Automation chapter for more info) or in situations where the GUI is not available. It can also be used in shell scripts or cron jobs to automatically gather information from a Gravwell instance, if needed. The intent is that the command line interface should provide every function available in the GUI.

12.1 Running the Client

The command-line client is installed as `/usr/sbin/gravwell` by default. It will attempt to connect to localhost unless a different server is specified with the "-s" flag, but be aware that since Gravwell's default install disables HTTPS, we'll need to use the "-insecure-no-https" flag. The client will prompt for a username and password; use the same credentials you would use to log in to the GUI:

```
/ # gravwell -insecure-no-https
Username:  admin
Password:  changeme
#>
```

Once you log in, the client will show a top-level prompt. Similar to the Cisco command line, the client implements multi-level menus; by typing 'admin', we can enter the administrative menu, then we can type 'help' to get a listing of available commands. Here is a snippet of available commands:

```
#>  admin
admin>  help
add_user            Add a new user
impersonate_user    Impersonate an existing users
del_user            Delete an existing user
get_user            Get an existing users details
update_user         Update an existing user
list_users          List all users
lock_user           Lock a user account
user_activity       Show a specific users activity
```

Help is available at all menu levels. When a command is selected, the client will prompt for input:

```
#>  search
query>  tag=* count
time range> -6h
count 1100
1/1
```

```
Press q[Enter] to quit, [Enter] to continue

Total Items: 1
1101 stats records from Mar 22 16:29:53.000 <-> Mar 22 22:29:54.000
count 1.10 K/1.00 663.35 KB/547 B 65.238537ms
```

12.2 Hands-on Lab: Basic CLI exploration

Launch a Gravwell webserver+indexer container:

```
docker run --rm --net gravnet -p 8080:80 -d --name gravwell gravwell:base
```

Now get a shell on the Gravwell container and run the client:

```
$ docker exec -it gravwell /bin/sh
/ # gravwell -insecure-no-https
Username:  admin
Password:  changeme
#>
```

Try the following tasks:

1. Create a new user
2. Lock that user
3. Verify that the user has been locked.
4. Unlock the user
5. Delete the user

When you're done, use the "exit" command to quit the CLI, then "exit" again to leave the shell in the Docker container.

Now, let's use the command line client non-interactively to run a search and download the results. First, we'll ingest 1,000 random JSON entries into the Gravwell instance:

```
docker run --net gravnet --rm -it --name ingesters gravwell:ingesters \
/opt/gravwell/bin/jsonGenerator -clear-conns gravwell:4023 -entry-count 1000
```

Now, we use the client to run a search in the background; the search will find all entries containing the name "David":

```
$ docker exec -it gravwell /bin/sh
/ # gravwell -insecure-no-https -b search
query>  tag=json grep David
time range> -6h
Background search with ID 569547375 launched
```

And use the client to download the results:

```
/ # gravwell -insecure-no-https download
+----------+-----+---------+----------+--------------------+-----------+
|Search ID |User |   Group |    State |   Attached Clients |   Storage |
+==========+=====+=========+==========+====================+===========+
|569547375 |   1 |       0 |  DORMANT |                  0 |  12.66 KB |
+----------+-----+---------+----------+--------------------+-----------+
search ID>  569547375
Available formats:
json
```

```
text
csv
format>  text
Save Location (default: /tmp)>  /tmp
Saving to  /tmp/569547375.text
```

Lab Questions

1. Download the results in csv and json format. What's the difference?
2. What happens when you leave out the "-b" flag while launching the search?

To clean up after the experiment, run:

```
docker kill $(docker ps -a -q)
```

Chapter 13

The Gravwell REST API

13.1 Introduction

Gravwell implements a REST API over HTTP. This API powers the Gravwell UI, but it can also be used to interface other systems with Gravwell. For instance, a Python script can run a Gravwell query by hitting a single API endpoint. This chapter discusses *API Tokens*, which are special authentication tokens given to client applications for accessing the API, and demonstrates how to access perhaps the most useful REST endpoint: the direct search API.

13.2 API Tokens

Gravwell users can generate tokens which allow an application to act as that user, but with limited privileges. Tokens are passed to the Gravwell webserver with HTTP requests as a method of authentication. Tokens are managed through the Tokens page (Figure 13.1), accessible in the Main Menu under the "Tools and Resources" section.

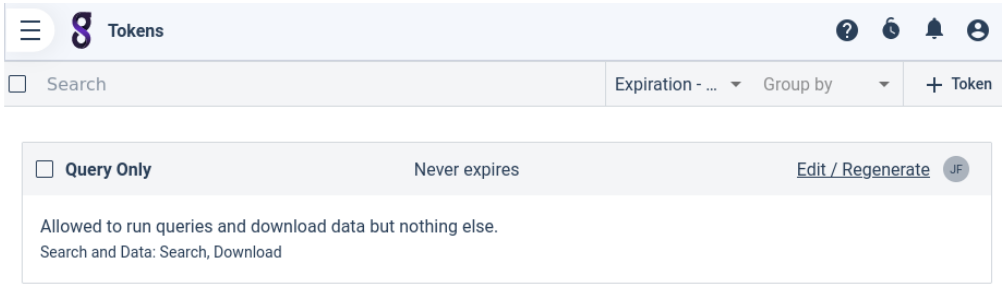


Figure 13.1: The token management page.

New tokens are created by clicking the "+ Token" button in the upper right. This brings up a page where the token's name and description may be specified and a list of permissions can be selected. Figure 13.2 demonstrates how fine-grained capabilities can be assigned; in this example, the token will be able to run searches, download the results, get a list of tags, and even ingest additional entries, but it will *not* be able to mark searches for saving or view search history. By carefully selecting permissions, it is possible to restrict the potential for harm if an API token is leaked.

Tokens can be set to expire after a period of time. By default, tokens last indefinitely, but the dropdown at the bottom of the new token page can be used to set a limited duration as seen in Figure 13.3.

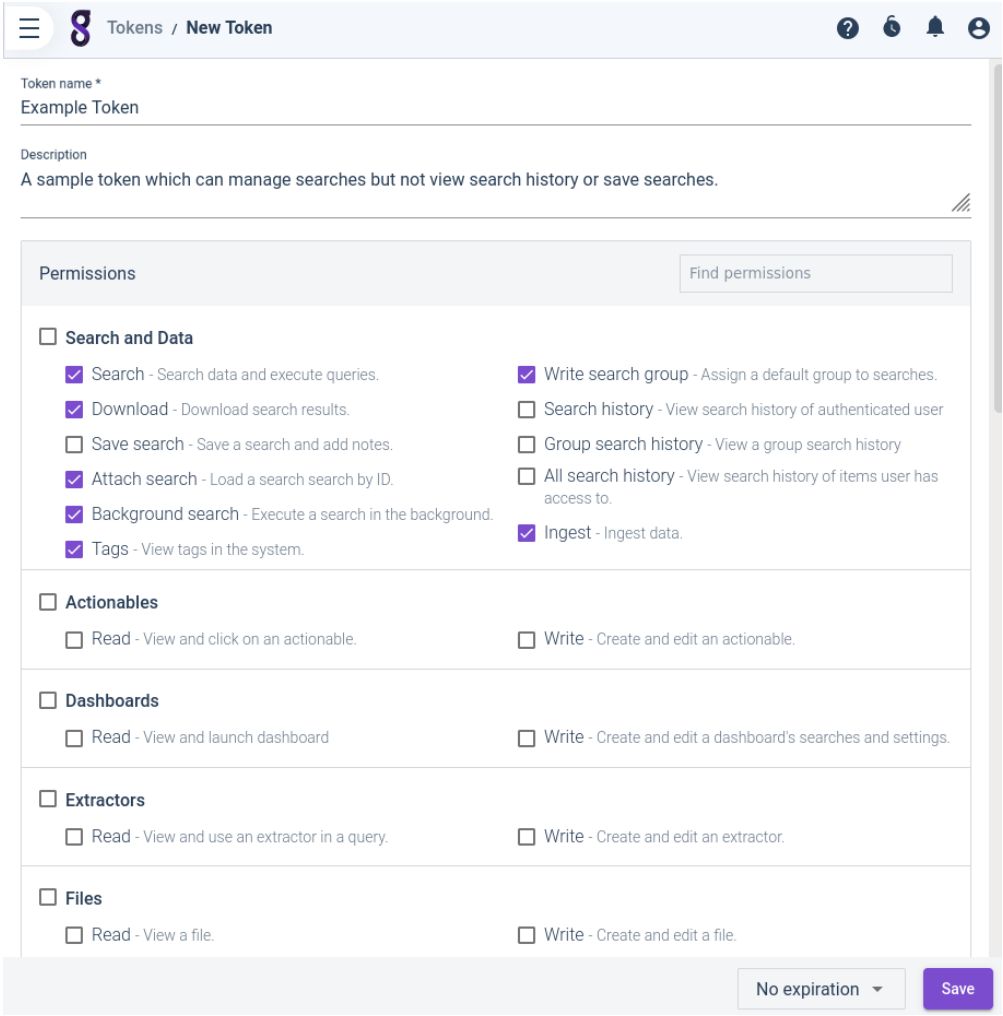


Figure 13.2: Defining a new token.

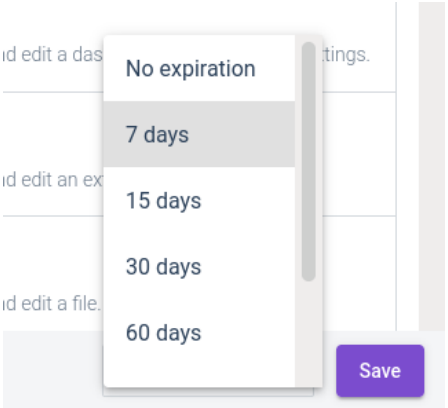


Figure 13.3: Token expiration options.

Once permissions have been selected for the new token, clicking “Save” will bring up a dialog showing the secret key for the token (Figure 13.4. This key should be copied at this time; it cannot be retrieved again later.

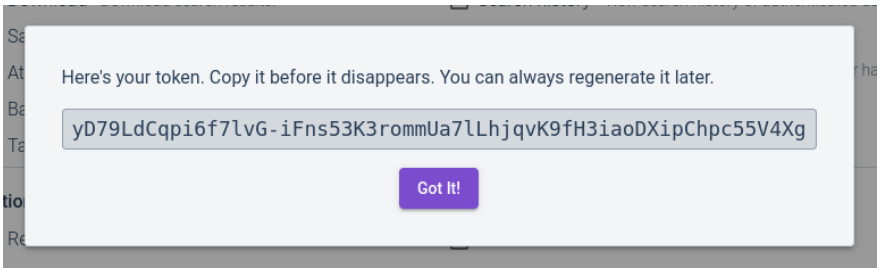


Figure 13.4: Token secret.

If the secret is lost, the token can be *regenerated*, creating a new secret key, but any applications using the old secret will stop working until updated.

13.2.1 Token Permissions and Restrictions

Token permissions are defined using specific allowances, in which the user selects exactly which functionality a given token is allowed to perform. The Gravwell user interface provides some nice features to let you select groups of permissions that might be logically related, but in the end each token must declare exactly what APIs and systems it is allowed to access. Most permissions are divided into read and write components. This means a token might be configured so it can read resources but not write them, or a token can read the state of automation scripts but not create, update, or schedule them.

Permissions on tokens are an overlay on the user’s existing permissions. This means that if the current user cannot access an API or feature, then the token cannot either–tokens can only restrict access, they cannot grant access that a user does not currently have.

Note: The ‘Token write’ permission can be particularly dangerous. If you grant a token the ability to write to the token API, it can create new tokens with any permission it wants. Token permissions are not transitive, so a restricted token with access to the ‘Token Write’ permission could create new tokens with no restrictions.

The token system only provides access to user-level APIs and cannot be used by admins to access admin-level APIs. An admin can still create and use tokens, but they cannot access APIs that require admin permissions. For example, an admin cannot use a token to create users or update a license.

13.3 Accessing the Gravwell API

There are many, many REST endpoints in the Gravwell API. A reasonably complete listing is available on the Gravwell documentation site.[1] The easiest way to interact with the API is using the `curl` command. Authentication is done by including a token secret (as described above) into a header named “Gravwell-Token”. For example, to fetch a list of tags in the system:

```
% curl -H 'Gravwell-Token: DQ-Tgpe56czTSZ_kzzDlFqqMKSkIR65hOoTmcdUK0J-7DMq3H2YmDK-0WQ' \
http://gravwell.example.org/api/tags
["default","syslog","kernel","dmesg","windows","www","netflow","ipfix"]
```

13.4 Direct Search API

The Gravwell Direct Query API is designed to provide atomic, REST-powered access to the Gravwell query system. This API allows for simple integration with external tools and systems that do not normally know

[1] https://docs.gravwell.io/#!api/api.md

how to interact with Gravwell. The API is designed to be as flexible as possible and support any tool that knows how to interact with an HTTP API.

The Direct Query API is authenticated and requires a valid Gravwell account with access to the Gravwell query system; a Gravwell token is the best way to access the API.

Issuing a query via the Direct Query API requires the same set of parameters as issuing a query via the Gravwell web GUI: a query string, a time range, and an optional output format. The Direct Query API has some limitations on which output formats can be provided. For example, the pointmap and heatmap renderers cannot output rendered maps via this API, nor can this API draw a chart and deliver it as an image. This API is primarily used for retrieving raw results and delivering them to other systems for direct integration.

The Direct Query API is atomic: one request will execute an entire search and deliver the completed results. Queries that cover large time ranges or require significant time to execute may require that HTTP clients adjust their respective client timeouts.

13.4.1 Query Endpoints

The Direct Query API consists of two REST endpoints which can parse a search and execute a search. The parse API is useful for testing whether a query is valid and could execute while the search API will actually execute a search and deliver the results. Both the query and parse APIs require the user and/or token to have the 'Search' permission.

Parse API

The parse API is used to validate the syntax of a Gravwell query before attempting to execute it. The parse API is accessed via a POST request to `/api/parse`. The parse API a query string delivered by header value, URL parameter, or a ParseSearchRequest[2] JSON object. A ParseSearchResponse[3] object and a 200 code will be returned if the query is valid.

The following curl commands are all functionally equivalent:

```
curl -X POST \
   -H "Gravwell-Token: aFOa_YbO7PeOMAqKO8PSD-oTrEZxopc5JBfOhuOW5_Vo-FxWsjHp" \
   -H "query: tag=gravwell limit 10" \
   http://10.0.0.1/api/parse
```

```
curl -X POST \
   -H "Gravwell-Token: aFOa_YbO7PeOMAqKO8PSD-oTrEZxopc5JBfOhuOW5_Vo-FxWsjHp" \
   http://10.0.0.1/api/parse?query=tag%3Dgravwell%20limit%2010
```

```
curl -X POST -H -d '{"SearchString":"tag=gravwell limit 10"}' \
   "Gravwell-Token: aFOa_YbO7PeOMAqKO8PSD-oTrEZxopc5JBfOhuOW5_Vo-FxWsjHp" \
   http://10.0.0.1/api/parse
```

An example response is shown below:

```
[H]
{
  "Sequence": 0,
  "GoodQuery": false,
  "ParsedQuery": "tag=gravwell limit 10",
  "RawQuery": "tag=gravwell limit 10",
  "ModuleIndex": 0,
  "CollapsingIndex": 1,
  "RenderModule": "text",
```

[2] https://pkg.go.dev/github.com/gravwell/gravwell/v3/client/types#ParseSearchRequest
[3] https://pkg.go.dev/github.com/gravwell/gravwell/v3/client/types#ParseSearchResponse

```
  "TimeZoomDisabled": false,
  "Tags": [
    "gravwell"
  ],
  "ModuleHints": [
    {
      "Name": "limit",
      "ProducedEVs": null,
      "ConsumedEVs": null,
      "ResourcesNeeded": null,
      "Condensing": true
    },
    {
      "Name": "limitCollapser",
      "ProducedEVs": null,
      "ConsumedEVs": null,
      "ResourcesNeeded": null,
      "Condensing": true
    },
    {
      "Name": "sort",
      "ProducedEVs": null,
      "ConsumedEVs": null,
      "ResourcesNeeded": null,
      "Condensing": false
    }
  ]
}
```


Query API

The query API actually runs a search and returns the results. It is accessed via a POST to `/api/search/direct`. The search API requires the parameters in Table 13.1 be delivered by header values, URL parameters, or a JSON object.

Parameter	Description	Optional
query	A complete Gravwell query string	
format	Query output format	
preview	Boolean indicating that the query should execute as a preview query	yes
start	RFC3339 start timestamp for the query	yes
end	RFC3339 end timestamp for the query	yes
duration	Go-encoded duration	yes

Table 13.1: Query Request Parameters

Note that while the 'start', 'end', and 'duration' parameters are optional at least one complete set must be provided, either 'start' and 'end' or 'duration'.

The results of the query are returned in the body of the response. Each query renderer will support a different set of output formats. If the specified output format is not supported a 400 BadRequest response will be returned.

The following are all valid invocations; the first example shows a sample result as well:

```
% curl -X POST \
```

```
   -H "Gravwell-Token: aFOa_YbO7Pe0MAqK08PSD-oTrEZxopc5JBf0hu0W5_Vo-FxWsjHp" \
   -H "query: tag=gravwell stats count" \
   -H "duration: 1h" \
   -H "format: text" \
   http://10.0.0.1/api/search/direct
count 14599
```

```
% curl -X POST \
   -H "Gravwell-Token: aFOa_YbO7Pe0MAqK08PSD-oTrEZxopc5JBf0hu0W5_Vo-FxWsjHp" \
   'http://10.0.0.1/api/search/direct?query=tag%3Dgravwell%20stats%20count&format=text&duration=1h'
```

```
% curl -X POST \
   -H "Gravwell-Token: aFOa_YbO7Pe0MAqK08PSD-oTrEZxopc5JBf0hu0W5_Vo-FxWsjHp" \
   -d '{"SearchString":"tag=gravwell stats count","SearchStart":"2022-03-01T12:00:00Z",
"SearchEnd":"2022-03-01T13:00:00Z","Format":"text"}' \
   http://10.0.0.1/api/search/direct
```

This example mixes some parameters in a JSON object, some as headers, and some as URL parameters:

```
% curl -X POST \
   -H "Gravwell-Token: aFOa_YbO7Pe0MAqK08PSD-oTrEZxopc5JBf0hu0W5_Vo-FxWsjHp" \
   -d '{"SearchString":"tag=gravwell limit 10"}'
   -H 'duration: 1h' \
   http://10.0.0.1/api/search/direct?format=text
```

This example fetches the results in CSV format:

```
% curl -X POST \
   -H "Gravwell-Token: aFOa_YbO7Pe0MAqK08PSD-oTrEZxopc5JBf0hu0W5_Vo-FxWsjHp" \
   -H "query: tag=gravwell syslog Appname | stats count by Appname | table Appname count" \
   -H "duration: 1h" \
   -H "format: csv" \
   http://10.0.0.1/api/search/direct
```

This example fetches the results as raw PCAP and stores the output in `/tmp/port80.pcap`:

```
% curl -X POST \
   -H "Gravwell-Token: aFOa_YbO7Pe0MAqK08PSD-oTrEZxopc5JBf0hu0W5_Vo-FxWsjHp" \
   -H "query: tag=packets packet tcp.Port==80 ipv4.IP==192.168.1.1 | pcap" \
   -H "duration: 1h" \
   -H "format: pcap" \
   --output /tmp/port80.pcap \
   http://10.0.0.1/api/search/direct
```

Chapter 14

Securing Gravwell

This chapter discusses security considerations for a Gravwell cluster. It will discuss indexer and webserver security, securing ingesters, and user authentication.

This chapter makes frequent references to **shared secrets**. These are strings used by Gravwell components to communicate with each other without user intervention. Shared secrets are never transmitted directly over the network; instead, clients and servers use a challenge-response algorithm to verify that they have the same shared secret. Because the shared secrets must be stored in plaintext to perform challenge-response, they are potentially vulnerable to users on the same machine and should therefore be protected. The shared secrets can be stored in three different ways:

1. Directly in `gravwell.conf`. This is the simplest, but care should be taken to ensure the file is only readable by the gravwell user.
2. As environment variables. If done correctly, this can help protect the secrets, but take note that setting the environment variables in a Docker command line (e.g. `'-e GRAVWELL_INGEST_AUTH=MySecret'`) can make them vulnerable to interception by other users. See the Gravwell online documentation[1] for a description of environment variables.
3. As a Docker secret. This is only available for systems running in Docker containers. See the Gravwell online documentation[2] for a description of how to pass Docker secret files to a Gravwell instance.

14.1 TLS/HTTPS

By default, Gravwell is configured to communicate via unencrypted channels. This means that communications between the webserver and the user, as well as communication between ingesters and indexers, will be visible to any adversary capable of intercepting traffic. As later sections will explain, Gravwell's built-in authentication methods ensure that the ingester's shared secrets will not be leaked, but an adversary could potentially capture passwords as users authenticate to the webserver, view entries as they are ingested, etc.

Switching over to TLS-encrypted communications will have two benefits: it will encrypt traffic to prevent eavesdropping, and it will allow components to verify that they are actually connecting to the correct system rather than an adversary's man-in-the-middle attack. The datastore component cannot be used without installing TLS certificates, so enabling TLS encryption is also a necessary part of configuring distributed frontends.

We strongly recommend installing a properly-signed TLS certificate on the Gravwell webserver and indexer before deploying Gravwell in any production environment. If a properly-signed certificate is unavailable, Gravwell can be deployed with a self-signed certificate, but every component will need to be configured to ignore invalid certificates, which makes the system vulnerable to man-in-the-middle attacks.

[1] https://docs.gravwell.io/#!configuration/environment-variables.md
[2] https://docs.gravwell.io/#!configuration/environment-variables.md

14.1.1 Installing a properly-signed TLS certificate

A properly-signed TLS certificate is the most secure way to access Gravwell. Browsers will automatically accept the certificate without complaint.

Obtaining a certificate is outside the scope of this documentation; consider either purchasing a certificate through one of the traditional providers or using LetsEncrypt[3] to obtain a free one.

To use the certificate, Gravwell must be told where the certificate and key files are. Assuming the files are installed at `/etc/certs/cert.pem` and `/etc/certs/key.pem`, edit `gravwell.conf` to uncomment and populate the `Certificate-File` and `Key-File` options:

```
Certificate-File=/etc/certs/cert.pem
Key-File=/etc/certs/key.pem
```

Note: These files must be readable by the "gravwell" user, but make sure other users can't read them.

To enable HTTPS on the webserver, change the `Web-Port` directive from 80 to 443, then comment out the `Insecure-Disable-HTTPS` directive. To enable TLS-encrypted ingester connections, find and uncomment the line `TLS-Ingest-Port=4024`. To enable HTTPS for the search agent, open `/opt/gravwell/etc/searchagent.conf`, comment out the `Insecure-Use-HTTP=true` line and change the port in the `Webserver-Address` line from 80 to 443.

Finally, restart the webserver, indexer, and search agent:

```
systemctl restart gravwell_webserver.service
systemctl restart gravwell_indexer.service
systemctl restart gravwell_searchagent.service
```

14.1.2 Install a self-signed certificate

Although it is not as secure as a proper TLS certificate, a self-signed certificate will ensure encrypted communication between users and Gravwell. By instructing browsers to trust the self-signed cert, it is also possible to avoid recurring warning screens.

First, we will generate a 1-year certificate in `/opt/gravwell/etc` using `gencert`, a program we ship with the Gravwell install:

```
cd /opt/gravwell/etc
sudo -u gravwell ../bin/gencert -h HOSTNAME
```

Make sure to replace HOSTNAME with either the hostname or the IP address of your Gravwell system. You can specify multiple hostnames or IPs by separating them with commas, e.g. `gencert -h gravwell.example.com,10.0.0.1,192.168.0.3`

Now, open `gravwell.conf` and uncomment the `Certificate-File` and `Key-File` directives. The defaults should point correctly to the two files we just created.

To enable HTTPS on the webserver, change the `Web-Port` directive from 80 to 443, then comment out the `Insecure-Disable-HTTPS` line. To enable TLS-encrypted ingester connections, find and uncomment the line `TLS-Ingest-Port=4024`.

To enable HTTPS for the search agent, open `/opt/gravwell/etc/searchagent.conf`, comment out the `Insecure-Use-HTTP=true` line and change the port in the `Webserver-Address` line from 80 to 443. With a self-signed certificate, you will also need to ensure that `Insecure-Skip-TLS-Verify=true` is set.

Finally, restart the webserver, indexer, and search agent:

```
systemctl restart gravwell_webserver.service
systemctl restart gravwell_indexer.service
systemctl restart gravwell_searchagent.service
```

[3] https://letsencrypt.org

14.2 Indexer Security

Gravwell indexers communicate with two components: ingesters, and webservers. Users never interact directly with indexers.

14.2.1 Authentication & Secrets

Ingesters and webservers authenticate to the indexer using shared secrets. Ingesters are authenticated using the `Ingest-Auth` secret while webservers use the `Control-Auth` secret. These shared secrets are stored in plaintext and used to perform challenge-response authentication. This means ingesters and webservers can auth to the indexer over plaintext channels without leaking the shared secrets, but it also means these secrets are vulnerable to attackers who have access to the filesystems of these systems.

14.2.2 Indexer-Webserver Communications

Communications between the indexer and the webserver take place over TCP port 9404 by default (specified by the `Control-Port` parameter). These connections *are not encrypted*. Indexers often have to transfer very large numbers of entries to the webserver during a search, and encryption would significantly slow the transfer. For this reason, indexers and webservers should communicate only over trusted networks. If the indexer-webserver traffic must traverse the Internet, we strongly recommend using a VPN or other tunnel to transport the traffic.

14.2.3 Indexer-Ingester Communications

Ingesters can connect to indexers using either encrypted or unencrypted connections. By default, indexers listen for cleartext ingest connections on TCP port 4023. If a TLS certificate is set up on the indexer, it can also listen for encrypted connections; by convention, indexers should listen for TLS ingest connections on TCP port 4024.

14.3 Webserver Security

Webservers communicate with user clients, indexers, the search agent, and optionally the datastore. As the gateway into the Gravwell system, they are often accessible from public networks and should therefore be protected carefully.

14.3.1 Authentication & Secrets

Webservers store the following shared secrets:

- `Control-Auth`, used to authenticate the webserver to indexers and the datastore.
- `Search-Agent-Auth`, used by the search agent to authenticate to the webserver.

They also maintain a database of Gravwell users, stored by default in `/opt/gravwell/etc/users.db`. Passwords in the user database are bcrypt hashed, but care should still be taken to prevent unauthorized system users from reading this file.

14.3.2 Webserver-Indexer Communication

Communications between the indexer and the webserver take place over TCP port 9404 by default (specified by the `Control-Port` parameter). These connections *are not encrypted*. Indexers often have to transfer very large numbers of entries to the webserver during a search, and encryption would significantly slow the transfer. For this reason, indexers and webservers should communicate only over trusted networks. If the indexer-webserver traffic must traverse the Internet, we strongly recommend using a VPN or other tunnel to transport the traffic.

14.3.3 Webserver-User Communication

Users access the webserver via their web browsers over either HTTP or HTTPS. Authentication consists of an HTTP POST request containing a username and password, meaning a user's password may be transmitted in plaintext if TLS is not configured on the webserver. Because of this, *no Gravwell webserver should be used over the Internet without first installing TLS certificates.*

14.3.4 Webserver-Search Agent Communication

The automated search agent connects to a Gravwell webserver over the same HTTP or HTTPS ports as a regular user, but it authenticates using a challenge-response algorithm and the search agent-specific shared secret. The search agent will default to unencrypted HTTP, which may be acceptable when the search agent is on the same system as the webserver, but if a search agent is installed on a separate system it should be configured to use HTTPS.

14.3.5 Webserver-Datastore Communication

When multiple webservers are configured in a Gravwell cluster, they coordinate via the datastore component. Communications between the webservers and the datastore include (hashed) user passwords, search histories, user-uploaded resources, and other potentially sensitive items. For this reason, Gravwell *strongly suggests* that communication between the webserver and the datastore be TLS-encrypted; configuring a TLS certificate as described in an earlier section is an important step in setting up a Gravwell cluster with multiple webservers.

14.4 Ingester security

Ingesters communicate only with indexers, using a shared ingest secret which can be stored via the methods described at the beginning of this chapter. As documented in the section on indexer security, ingesters may communicate with the indexers over either cleartext or encrypted connections. In either case, the shared secret is never transmitted, so it is not susceptible to interception, but an adversary could potentially intercept *entries* as they are uploaded over a cleartext connection. For this reason we strongly recommend configuring TLS certificates on the indexer if you intend to run ingesters outside your trusted networks.

If all ingesters use the same ingest secret, and one is compromised, the attacker could theoretically use that compromised ingest secret to upload entries to your indexers. Consider deploying Federators (see Section 7.9) in order to segment your ingesters into different security enclaves to help mitigate this risk.

14.5 User Authentication and Lockout

As mentioned in the webserver section, users are authenticated to the webserver by sending their username and password in an HTTP POST request. If the webserver is not configured to offer HTTPS connections, user credentials are at risk of interception. Always set up TLS certificates on the webserver and enable HTTPS before allowing users to log on from public networks.

Gravwell provides a basic form of brute-force protection. If several failed login attempts are made over a short period, the webserver will delay its response to subsequent attempts by several seconds. This is intended to slow down brute-forcers while still allowing legitimate users to log in.

Index