Sagan User Guide Documentation

Release 1.2.2

Champ Clark III

Nov 11, 2022

Contents

CHAPTER 1

What is Sagan?

Sagan is a log analysis engine. It was designed with a Security Operations Center (SOC) in mind. This makes Sagan's operations different from most log analysis tools. Sagan is designed and meant to analyze logs across many different platforms in many different locations. A driving principle behind Sagan is for it to do the "heavy lifting" analysis before putting the event in front of a human. Another driving principle is to do all analysis of logs in "real time". This is also a differentiating factor of Sagan. In a SOC environment, waiting for hours for analysis simply isn't an option. Delaying analysis gives an attacker an advantage in that they will have been in your network undetected during that lag time. If you are a security professional reading this, you likely understand the real-time aspects of packet analysis. For example, security professionals would never accept prolonged delays in our Intrusion Detection and Intrusion Prevention engines. Nor would reasonable security professionals find it acceptable to analyze packet data the next day for security related events. With this in mind, we demand our packet analysis engines to work in real time or close to it. This premise is how projects like Snort (https://snort.org) and Suricata (https://suricata-ids.org) function.

Sagan treats log data similar to how IDS or IPS treats packet data. In fact, Sagan treats the data so similarly, that Sagan rules can confuse even the most seasoned security professionals.

1.1 License

Sagan is licensed under the GNU/GPL version 2.

CHAPTER 2

Installation

Before Sagan can be used it has to be installed. Sagan can be installed on various distributions using binary packages; however, these are typically out of date. Check your distribution to verify if the latest version of Sagan is available.

For people familiar with compiling their own software, the Source method is recommended.

2.1 libpcre (Regular Expressions)

Sagan uses `libpcre` to use 'Perl Compatible Regular Expressions'. This is used in many Sagan signatures and is a required dependency.

To install `libpcre` on Debian/Ubuntu:

```
sudo apt-get install libpcre3-dev libpcre3
```

To install `libpcre` on Redhat/CentOS:

```
sudo yum install pcre-devel
```

To install `libpcre` on FreeBSD/OpenBSD:

```
cd /usr/ports/devel/pcre && make && sudo make install
```

To install `libpcre` on Gentoo:

```
emerge -av libpcre
```

2.2 libyaml (YAML configuration files)

Sagan uses `libyaml` to read in configurations files. This is a required dependency.

To install `lbyaml` on Debian/Ubuntue:

```
apt-get install libyaml-dev
```

To install `libyaml` on Redhat/CentOS:

```
yum install libyaml-devel
```

To install `libyaml` on FreeBSD/OpenBSD:

```
cd /usr/ports/textproc/libyaml/ && sudo make install
```

To install `libyaml` on Gentoo:

```
emerge -av libyaml
```

2.3 Other dependencies

While `libpcre` and `libyaml` are required Sagan dependencies, you'll likely want Sagan to perform other functions like parsing JSON data or writing data out in various formats. While these prerequisites are not required, you should look them over for further functionality.

2.4 liblognorm (Normalization)

While not a required dependency, it is recommended that you install liblognorm. This library can be used by Sagan to extract useful data from incoming log data. `liblognorm` is part of the `rsyslog` daemon. Note: Installing `liblognorm` will automatically install `libfastjson`.

More information about `liblognorm` can be found at the *LibLogNorm <https://FIXME>*_ web site.

To install `liblognorm` on Debian/Ubuntu:

```
apt-get install liblognorm-dev liblognorm5
```

To install `liblognorm` on Redhat/Centos:

```
yum install liblognorm
```

To build `liblognorm` from source code, see ADD THIS IN

2.5 libfastjson (JSON)

If you install `liblognorm`, you do not need to install `libfastjson` as it is part of the `liblognorm` package. The library is a fork of `json-c` by the `rsyslog` team. It has improvements which make parsing and building JSON data faster and more efficent.

To install `libfastjson` on Debian/Ubuntu:

```
apt-get install libfastjson-dev libfastjson4
```

To install `liblfastjson` on Redhat/Centos:

```
LOOK THIS UP
```

To install `libfastjson` on FreeBSD/OpenBSD:

```
LOOK THIS UP
```

To install `libjson` on Gentoo:

```
LOOK THIS UP
```

To build `libjson` from source code, see ADD THIS IN

2.6 libesmtp (SMTP)

Sagan has the ability as an `output-plugin` to send alerts via e-mail. If you would like this type of functionality, you will need to install `libesmtp`.

To install `libesmtp` on Debian/Ubuntu:

```
apt-get install libesmtp-dev
```

To install `libesmtp` on FreeBSD/OpenBSD:

```
cd /usr/ports/mail/libesmtp && make && sudo make install
```

To install `libesmtp` on Gentoo:

```
emerge -av libesmtp
```

2.7 libmaxminddb (GeoIP)

Sagan can do GeoIP lookups of Internet Addresses. Rules that use this functionality are part of the `-geoip.rules` rule sets. While not required, the data can be very useful.

To install `libmaxminddb` on Debian/Ubuntu:

```
apt-get install libmaxminddb0 libmaxminddb-dev geoip-database-contrib geoipupdate
```

To install `libmaxminddb` on Redhat/CentOS:

```
yum install GeoIP GeoIP-devel GeoIP-data
```

From time to time you will need to update your MaxMind GeoIP Lite Databases . Typcially, you'll need to do something like this:

Basic Maxmind GeoIP2 Country Code updates:

```
cd /usr/local/share/GeoIP2
sudo wget http://geolite.maxmind.com/download/geoip/database/GeoLite2-Country.tar.gz
sudo gzip -d GeoLite2-Country.tar.gz
```

2.8 hiredis (Redis)

Sagan has the ability to store `flexbit` data in a Redis database. This allows data to be shared over a distributed enviornment. **This feature is considered beta**. To use this functionality you will need to install the `hiredis` library.

To install `hiredis` on Debian/Ubuntu:

```
apt-get install libhiredis-dev
```

To install `hiredis` on Redhat/CentOS:

```
sudo yum install redis
```

To install `hiredis` from source, see the Hiredis Github Page .

2.9 libpcap (Sniffing logs)

By using the `libpcap` library, Sagan has the ability to 'sniff' unencrypted logs 'off the wire' and process them. This can be useful for capturing logs in transit to a centralized log server. It can also be useful for testing Sagan's effectiveness before doing a full deployment. You will need a method to 'capture' the traffic off the wire. This is typically done via a `span` port or a `network tap`.

To install `libpcap` on Debian/Ubuntu:

```
apt-get install libpcap-dev
```

To install `libpcap` on Redhat/CentOS:

```
yum install libpcap
```

To install `libpcap` on Gentoo:

```
emerge -av libpcap
```

CHAPTER 3

Compiling Sagan

Installation from source distributions files.

Basic steps:

```
git clone https://github.com/quadrantsec/sagan.git
cd sagan
./autogen.sh
./configure
make
sudo make install
```

By default, Sagan builds with the `--enable-lognorm` (See `liblognorm` above) option enabled. Any other options need to be manually enabled or disabled.

3.1 Quick start from source

The first example installs Sagan with the basics (all prerequisites and `liblognorm`).

Quick start with the bare basics:

```
sudo apt-get install libpcre3-dev libpcre3 libyaml-dev liblognorm-dev
wget https://quadrantsec.com/download/sagan-current.tar.gz
cd sagan-1.2.1
./configure
make
sudo make install
```

This example Quick start installs Sagan with more features including the required prerequisites, `libognorm` (log normalization), `libesmtp` (e-mail support), `libmaxminddb` (GeoIP), `hiredis` (Redis), `libpcap` (sniffing logs).

3.2 A more complete quick start

This example installs Sagan with the most common and useful prerequisites.

A more complete quick start:

```
sudo apt-get install build-essential libpcre3-dev libpcre3 libyaml-dev liblognorm-dev␣
↪libesmtp-dev libmaxminddb0 libmaxminddb-dev libhiredis-dev libpcap-dev liblognorm-
↪dev libfastjson-dev libestr-dev pkg-config zlib1g-dev
wget https://quadrantsec.com/download/sagan-2.0.1.tar.gz
tar -xvzf sagan-2.0.1.tar.gz
cd sagan-2.0.1
./configure --enable-geoip --enable-esmtp --enable-libpcap --enable-redis --enable-
↪gzip
make
sudo make install
```

3.3 Prerequisites

Before compiling and installing Sagan, your system will need some supporting libraries installed. The primary prerequisites are `libpcre`, `libyaml` and `libpthreads` (note: most systems have `libpthread` installed by default). While there are no other required dependencies other than these, you should look over the others for expanded functionality. For example, `liblognorm` **is not required but highly recommended**.

3.4 Common configure options

`--prefix``=/usr/`
Installs the Sagan binary in the /usr/bin. The default is `/usr/local/bin`.

`--sysconfdir``=/etc`
Installs the Meer configuration file (meer.yaml) in the /etc directory. The default is `/usr/local/etc/`.

`--with-libyaml_libraries`
This option points Sagan to where the libyaml files reside.

`--with-libyaml-includes`
This option points Sagan to where the libyaml header files reside.

`--disable-snortsam`
This option disables *Snortsam <http://www.snortsam.net/>*_ support. Snortsam is a firewall blocking agent for Snort.

`--enable-esmtp`
This option enabled Sagan's ability to send data and alerts via e-mail. In order to use this functionality, you will need `libesmtp` support (see above).

`--with-esmtp-includes``=DIR`
This points `configure` to the libesmtp header files (see `--enable-esmtp`).

`--with-esmtp-libraries``=DIR`
This points `configure` to the library location of `libesmtp` (see `--enable-esmtp`).

`--enable-geoip`
This option allows Sagan to do GeoIP lookups of TCP/IP addresses via the Maxmind GeoIP2 Lite to determine countries of origin or destination.

--with-geoip-includes=DIR
: This points `configure` to the Maxmind GeoIP header data (see `--enable-geoip`).

--with-geoip-libraries=DIR
: This points `configure` to the Maxmind GeoIP library location (see `--enable-geoip`).

--disable-syslog
: By default, Sagan can send alerts to syslog. This option disables this feature.

--enable-system-strstr
: By default, Sagan uses a built in assembly version of the C function `strstr()` for rule `content` checks. This code is CPU specific and may cause issues on non-x86 hardware. This option disables Sagans built in `strstr` and uses the default operating system's `strstr`. This option is useful when building Sagan on embedded systems.

--enable-redis
: Sagan has the ability to store `flexbits` in a Redis database. This option enables this Redis feature. You need the `libhiredis` library installed (see `libhiredis` above).

--disable-lognorm
: Sagan uses `liblognorm` to 'normalize' log data. This disables that feature.

--with-lognorm-includes=DIR
: Points `configure` to the liblognorm header files.

--with-lognorm-libraries=DIR
: Points `configure` to the liblognorm library.

--enable-libpcap
: This option enables Sagan to 'sniff' logs off the network. The `libpcap` library needs to be installed (see `libpcap` above).

--with-libpcap-includes=DIR
: Points `configure` to the `libpcap` header files.

--with-libpcap-libraries=DIR
: Points `configure` to the `libpcap` library directory (see `libpcap` above).

--disable-libfastjson
: This option disables processing and producting JSON output. Note: Using `liblognorm` automatically enables this feature. **You probably don't want to do with**

--with-libfastjson-includes=DIR
: Points `configure` to the `libfastjson` header files.

--with-libfastjson-libraries=DIR
: Points `configure` to the `libfastjson` library directory.

--enable-bluedot
: Bluedot is <Quadrant Information Security's <https://quadrantsec.com>`_ 'Threat Intelligence' plateform. This allows Sagan to perform lookups of TCP/IP addresses, file hashes, etc. **Note: You likely do not need this option as the API is not publically available at this time**.

--with-libpthread-includes=DIR
: Points `configure` to the `libpthread` header files.

--with-libpthread-libraries=DIR
: Points `configure` to the `libpthread` library directory.

--with-libyaml-includes=DIR
: Points `configure` to the `libyaml` header files.

--with-libyaml-libraries=DIR
Points `configure` to the `libyaml` library directory.

--with-libpcre-includes=DIR
Points `configure` to the `libpcre` header files.

--with-libpcre-libraries=DIR
Points `configure` to the `libpcre` library directory.

3.5 Post-installation setup and testing

Create a "sagan" user and related directories:

```
sudo useradd --system -d /var/sagan -s /bin/false sagan
sudo mkdir -p /var/sagan/fifo /var/log/sagan /var/run/sagan
sudo mkfifo /var/sagan/fifo/sagan.fifo
sudo chmod 420 /var/sagan/fifo/sagan.fifo
sudo chown -R sagan:sagan /var/sagan /var/log/sagan /var/run/sagan
```

Checkout the "sagan-rules" repository into `/usr/local/etc/sagan-rules`:

```
cd /usr/local/etc
sudo git clone https://github.com/beave/sagan-rules
```

To test, run `sagan --debug syslog,engine` as the root user. It will switch to the sagan user when ready, and remain running in the foreground.

Manually generate a test syslog message in "pipe" format:

```
echo "192.0.2.1|local0|info|info|sshd|2001-01-01|00:00:00|sshd| User ubuntu not␣
↪allowed because shell /etc/passwd is not executable" |
  sudo tee /var/sagan/fifo/sagan.fifo
```

From the sagan process, you should see the syslog message received and rules triggered:

```
[D] [processor.c, line 168] **[Parsed Syslog]*********************************
[D] [processor.c, line 169] Host: 192.0.2.1 | Program: sshd | Facility: local0 |␣
↪Priority: info | Level: info | Tag: sshd | Date: 2001-01-01 | Time: 00:00:00
[D] [processor.c, line 170] Parsed message: User ubuntu not allowed because shell /
↪etc/passwd is not executable
[D] [processors/engine.c, line 1543] **[Trigger]*********************************
[D] [processors/engine.c, line 1544] Program: sshd | Facility: local0 | Priority:␣
↪info | Level: info | Tag: sshd
[D] [processors/engine.c, line 1545] Threshold flag: 0 | After flag: 0 | Flexbit␣
↪Flag: 0 | Flexbit status: 0
[D] [processors/engine.c, line 1546] Triggering Message: User ubuntu not allowed␣
↪because shell /etc/passwd is not executable
[D] [processors/engine.c, line 1543] **[Trigger]*********************************
[D] [processors/engine.c, line 1544] Program: sshd | Facility: local0 | Priority:␣
↪info | Level: info | Tag: sshd
[D] [processors/engine.c, line 1545] Threshold flag: 0 | After flag: 0 | Flexbit␣
↪Flag: 0 | Flexbit status: 0
[D] [processors/engine.c, line 1546] Triggering Message: User ubuntu not allowed␣
↪because shell /etc/passwd is not executable
```

The alert data is written to `/var/log/sagan/alert.log`:

```
[**] [1:5000020:4] [OPENSSH] Not executable shell - login attempt [**]
[Classification: unsuccessful-user] [Priority: 1] [192.0.2.1]
[Alert Time: 10-28-2019 15:25:44.584658]
2001-01-01 00:00:00 192.0.2.1:514 -> 192.0.2.1:22 local0 info sshd
Message: User ubuntu not allowed because shell /etc/passwd is not executable
[Xref => http://wiki.quadrantsec.com/bin/view/Main/5000020]

[**] [1:5000077:3] [OPENSSH] Attempt to login using a denied user [**]
[Classification: unsuccessful-user] [Priority: 1] [192.0.2.1]
[Alert Time: 10-28-2019 15:25:44.584658]
2001-01-01 00:00:00 192.0.2.1:514 -> 192.0.2.1:22 local0 info sshd
Message: User ubuntu not allowed because shell /etc/passwd is not executable
[Xref => http://wiki.quadrantsec.com/bin/view/Main/5000077]
```

Notice that this particular message triggers two rules - you can find them both in `/usr/local/etc/sagan-rules/openssh.rules` by searching for the rule IDs.

Finally, configure the system to run the daemon in the background. Create `/etc/systemd/system/sagan.service` containing:

```
[Unit]
Description=Sagan daemon
Documentation=https://sagan.readthedocs.io/
Before=rsyslog.service syslog-ng.service

[Service]
User=sagan
Group=sagan
EnvironmentFile=-/etc/default/sagan
ExecStart=/usr/local/bin/sagan $OPTIONS
ExecReload=/bin/kill -HUP $MAINPID
Restart=on-failure

[Install]
WantedBy=multi-user.target
```

Then load and start it:

```
sudo systemctl daemon-reload
sudo systemctl start sagan
sudo systemctl enable sagan
```

CHAPTER 4

Command Line Option

This document needs to be completed!

CHAPTER 5

Syslog Configuration

Sagan typically receives its data from a third party daemon. This is typically something like `rsyslog`, `syslog-ng` or `nxlog`. The first step is to get one of those systems set up.

5.1 rsyslog - "pipe" mode

Below is a simple rsyslog configuration to output to Sagan in a legacy "pipe" delimited format. The Sagan `input-type` (set in the `sagan.yaml`) will need to be set to `pipe`. For more information, consult the rsyslog documentation for templates, properties and the property replacer.

Example `rsyslog` "pipe" configuration, can be installed as `/etc/rsyslog.d/10-sagan.conf`:

```
template(name="SaganPipe" type="list") {
    property(name="fromhost-ip")
    constant(value="|")
    property(name="syslogfacility-text")
    constant(value="|")
    property(name="pri")
    constant(value="|")
    property(name="syslogseverity-text")
    constant(value="|")
    property(name="syslogtag")
    constant(value="|")
    property(name="timereported" dateformat="year")
    constant(value="-")
    property(name="timereported" dateformat="month")
    constant(value="-")
    property(name="timereported" dateformat="day")
    constant(value="|")
    property(name="timereported" dateformat="hour")
    constant(value=":")
    property(name="timereported" dateformat="minute")
    constant(value=":")
```

(continues on next page)

(continued from previous page)

```
    property(name="timereported" dateformat="second")
    constant(value="|")
    property(name="programname")
    constant(value="|")
    # Note: already escaped if EscapecontrolCharactersOnReceive is on (default)
    property(name="msg" controlcharacters="escape")
    constant(value="\n")
}

*.*  action(type="ompipe" pipe="/var/sagan/fifo/sagan.fifo" template="SaganPipe")
```

NOTE: rsyslog's "msg" property includes the space after the colon. This is important because Sagan's liblognorm rules also expect the leading space.

To receive over UDP you'll also need to uncomment these lines in `/etc/rsyslog.conf`:

```
# provides UDP syslog reception
module(load="imudp")
input(type="imudp" port="514")
```

Set appropriate permissions on the fifo before restarting rsyslog. (Beware: if you run sagan as root, it will reset them back again):

```
sudo chown sagan:syslog /var/sagan/fifo/sagan.fifo
sudo chmod 420 /var/sagan/fifo/sagan.fifo
sudo systemctl restart rsyslog
```

To test:

```
logger -t sshd "User ubuntu not allowed because shell /etc/passwd is not executable"
```

5.2 rsyslog - JSON mode

Below is a simple rsyslog configuration to output to Sagan in a "JSON" format. The Sagan `input-type` (set in the `sagan.yaml`) will need to be set to `json`. You will also need to set your `json-software` to `rsyslog`.

This uses rsyslog's standard JSON output:

```
template(name="SaganJson" type="list") {
    property(name="jsonmesg")
    constant(value="\n")
}

*.*  action(type="ompipe" pipe="/var/sagan/fifo/sagan.fifo" template="SaganJson")
```

It formats messages as per the following sample:

```
{
    "msg": " Stopping System Logging Service...",
    "rawmsg": "<30>Oct 28 16:32:13 systemd[1]: Stopping System Logging Service...",
    "timereported": "2019-10-28T16:32:13.970608+00:00",
    "hostname": "sagan",
    "syslogtag": "systemd[1]:",
    "inputname": "imuxsock",
```

(continues on next page)

(continued from previous page)

```
    "fromhost": "sagan",
    "fromhost-ip": "127.0.0.1",
    "pri": "30",
    "syslogfacility": "3",
    "syslogseverity": "6",
    "timegenerated": "2019-10-28T16:32:13.970608+00:00",
    "programname": "systemd",
    "protocol-version": "0",
    "structured-data": "-",
    "app-name": "systemd",
    "procid": "1",
    "msgid": "-",
    "uuid": null,
    "$!": null
}
```

Unfortunately it does not include the text versions of the facility and severity, nor format the date and time the way Sagan expects. So an alternative approach is to build up the JSON message explicitly containing the required fields:

```
template(name="SaganJson" type="list") {
    constant(value="{")
    property(name="fromhost-ip" format="jsonf")
    constant(value=",")
    property(name="syslogfacility-text" format="jsonf")
    constant(value=",")
    property(name="pri" format="jsonf")
    constant(value=",")
    property(name="syslogseverity-text" format="jsonf")
    constant(value=",")
    property(name="syslogtag" format="jsonf")
    constant(value=",\"date\":\"")
    property(name="timereported" dateformat="year")
    constant(value="-")
    property(name="timereported" dateformat="month")
    constant(value="-")
    property(name="timereported" dateformat="day")
    constant(value="\",\"time\":\"")
    property(name="timereported" dateformat="hour")
    constant(value=":")
    property(name="timereported" dateformat="minute")
    constant(value=":")
    property(name="timereported" dateformat="second")
    constant(value="\",")
    property(name="programname" format="jsonf")
    constant(value=",")
    property(name="msg" format="jsonf")
    constant(value="}\n")
}

*.*  action(type="ompipe" pipe="/var/sagan/fifo/sagan.fifo" template="SaganJson")
```

To use this, set your `json-software` to `rsyslog-alt`.

5.3 syslog-ng - "pipe" mode

Below is a simple Syslog-NG configuration to output to Sagan in a legacy "pipe" delimited format. For more complex configurations, please consult the `syslog-ng` documentation. The Sagan `input-type` (set in the `sagan.yaml`) will need to be set to `pipe`.

Example `syslog-ng` "pipe" configuration:

```
# Sources of log data.

source s_src { system(); internal(); };       # Internal
source syslog_in { udp(port(514)); };         # UDP port 514

# A "destination" to send log data to.  In our case, a named pipe (FIFO)

destination sagan_fifo {
  pipe("/var/sagan/sagan.fifo"
  template("$SOURCEIP|$FACILITY|$PRIORITY|$LEVEL|$TAG|$YEAR-$MONTH-$DAY|$HOUR:$MIN:
↪$SEC|$PROGRAM| $MSG\n")
  template-escape(no)
  );
};

# This line ties the sources and destinations together.

log { source(s_src); destination(sagan_fifo); };
log { source(syslog_in); destination(sagan_fifo); };
```

5.4 syslog-ng - JSON mode

Below is a simple Syslog-NG configuration to output to Sagan in a "JSON" format. For more complex configurations, please consult the `syslog-ng` documentation. The Sagan `input-type` (set in the `sagan.yaml`) will need to be set to `json`. You will also need to set your `json-software` to `syslog-ng`.

Using the Sagan JSON format allows for more flexibility with the log data and is recommended.

Example `syslog-ng` JSON configuration:

```
# Sources of log data.

source s_src { system(); internal(); };       # Internal
source syslog_in { udp(port(514)); };         # UDP port 514

# A "destination" to send log data to.  In our case, a named pipe (FIFO)

destination sagan_fifo {
      pipe("/var/sagan/sagan.fifo"
      template("$(format-json --scope selected_macros --scope nv_pairs)\n"));
      };

# This line ties the sources and destinations together.

log { source(s_src); destination(sagan_fifo); };
log { source(syslog_in); destination(sagan_fifo); };
```

5.5 nxlog

5.6 other sources

CHAPTER 6

Sagan Configuration

The primary Sagan configuration file is `sagan.yaml`. Its default location is the `/usr/local/etc` directory.

Comments within the `sagan.yaml` file start with a '#'. Stand-alone comments (on lines of their own) and comments after statements are valid.

The `sagan.yaml` is broken up in several parts. Those parts are `vars`, `sagan-core`, `processors`, `outputs` and `rule-files`.

6.1 Sagan with JSON input

Sagan reads data from your favorite syslog daemon (rsyslog, syslog-ng, nxlog, etc) via a "named pipe" (also known as a FIFO). A named pipe operates similarly to a file but with the writer (your syslog daemon) and a reader (Sagan). Rather than the contents being written to a disk or file, the data is stored in kernel memory. This data will wait in kernel memory until a process (Sagan) reads it. Named pipes (FIFOs) allow for separate processes to communicate with each other. Since this happens in kernel memory, the communications is extremely fast.

In order for the writer (syslog daemon) and reader (Sagan) to be able to share data, there has to be a standard between the two. Traditionally, Sagan required the syslog daemon to write data to the file in a very specific format. This was done by a delimiting the data via the '|' (pipe) symbol. This format was similar to a CSV file.

A newer and more flexible way for the writer (syslog daemon) and reader (Sagan) to share data is via JSON. Many modern day syslog daemons offer a JSON output format. This is the ideal method of sharing data as it allows the data to be more dynamic.

Sagan-core configurations for JSON

In the `sagan-core` section, in the sub section `core` is where you can set the `input-type`. There are two valid options. The legacy `pipe` format or `json`. If you are using the legacy `pipe` format, as long as both the syslog daemon can write to the named pipe in the proper format (see `Syslog Configuations`), there are no other configurations.

If you want to use the `input-type` of `json`, you'll need to specify the mapping type. Below is an example section of the `input-type`

```
input-type: json                          # pipe or json
json-map: "$RULE_PATH/json-input.map"     # mapping file if input-type: json
json-software: syslog-ng                  # by "software" type.
```

The `json-map` is a mapping file to assist Sagan in decoding JSON supplied by your syslog daemon. The `json-software` configures Sagan "what" JSON map to use in the `json-map`.

For example, let's say your syslog daemon is Syslog-NG configured to send JSON to the named pipe (JSON). The data going into the pipe might look similar to this:

```
{"TAGS":".source.s_src","SOURCEIP":"127.0.0.1","SEQNUM":"3341","PROGRAM":"sshd",
↪"PRIORITY":"info","PID":"23233","MESSAGE":"Failed password for root from 218.92.0.
↪190 port 34979 ssh2","LEGACY_MSGHDR":"sshd[23233]: ","HOST_FROM":"dev-2","HOST":
↪"dev-2","FACILITY":"auth","DATE":"Apr  3 03:00:46"}
```

Sagan needs to be able to identify the fields within the Syslog-NG formated JSON data. Within the `json-map` file, we have this line:

```
{"software":"syslog-ng","syslog-source-ip":"SOURCEIP","facility":"FACILITY","level":
↪"PRIORITY","priority":"PRIORITY","time":"DATE","date":"DATE","program":"PROGRAM",
↪"message":"MESSAGE"}
```

This maps the Syslog-NG fields to internal fields for Sagan to understand. For example, Sagan expects a "message" field. Syslog-NG has this field named "MESSAGE". This mapping maps "message" = "MESSAGE". Sagan's internal "syslog-source-ip" is mapped the Syslog-NG "SOURCEIP" field, and so on.

Take special note of the "software" at the beginning of the JSON input mapping file. This is the name of the "mapping" which is set in the `sagan.yaml`. In our example, the `json-software` field is set to `syslog-ng`. The mapping file contains mappings for multiple software types (syslog-ng, rsyslog, nxlog, etc). The `json-software` tells Sagan which mapping you want to use.

6.1.1 Sagan JSON variables

`"software": "{software type}"`
    This is the name of the mapping. This is used in the Sagan YAML `json-software` type.

6.1.2 Mappings:

`"syslog-source-ip"`
    TCP/IP address of where the log originated from. Typically the syslog server.

`"facility"`
    Syslog facility.

`"level"`
    Syslog level.

`"priority"`
    Syslog priority.

`"time"`
    Syslog timestamp.

`"date"`
    Syslog date.

`"message"`
Syslog "message" field. This is the only required option.

CHAPTER 7

vars

The var section of the sagan.yaml is a place reserved for declaring variables for the Sagan system to use. Using variables can be useful when you have multiple rules that use semi dynamic content. For example, let's say you have a signature that looks for a combination of users. In the vars area, you might set up a variable like this:

```
USERSNAME "bob, frank, mary, david"
```

Within a signature, you would then reference $USERNAME to have access to the values in that variable. If at a later date you wish to add or remove values from that variable, all signatures will adopt the new variable's values.

Variables can also be used within the sagan.yaml file. For example, when you set the RULE_PATH variable, it can be used within signatures but also within the sagan.yaml. By doing this, it allows you one location to make changes across multiple configuration options or signatures.

The vars section of the sagan.yaml is broken into subsections. These subsections are sagan-groups, address-groups, port-groups, geoip-groups, aetas-groups, mmap-groups, misc-groups. Each group has its own purpose and function. In the majority of cases, if you want to define variables of your own, you would put them in the misc-groups subsection.

The sagan-groups section is reserved for core Sagan function. For example, where to store lock files, where the FIFO (named pipe) is located for Sagan to read data from, where to store logs, etc.

Example sagan-groups subsection:

```
vars:

   # 'Core' variables used by Sagan.

   sagan-groups:

     FIFO: "/var/sagan/fifo/sagan.fifo"
     RULE_PATH: "/usr/local/etc/sagan-rules"
     LOCKFILE: "/var/run/sagan/sagan.pid"
     LOG_PATH: "/var/log/sagan"
```

The address-groups is an area to define your network. This is where you define values like $HOME_NETWORK and $EXTERNAL_NETWORK. In the majority of cases, you'll likely want to leave these any and any. You can create your own separate network groups here. For example, you could create a new variable `INTERNAL_NETWORK`. Addresses in this group are in the standard CIDR network notation. For example:

```
INTERNAL_NETWORK [10.0.0.0/8, 192.168.0.0/16]
```

Example `address-groups` subsection:

```
# HOME_NET and EXTERNAL_NET function similar to Suricata/Snort.  However,
# it's rare you'll want to set them.  In most situations leaving it set
# to "any" is best.

address-groups:

   HOME_NET: "any"
   EXTERNAL_NET: "any"
```

The `port-groups` is an area to define common ports and protocols. This section allows you to tailor ports used within your organization. For example, you might run SSH port TCP port 2222 rather than port 22. If you modified the variable in this section, it will be adopted by the rest of the rules.

Example `port-groups` subsection:

```
# Common ports used by common protocols.  These variables are used by
# rule sets.

port-groups:

  SSH_PORT: 22
  HTTP_PORT: 80
  HTTPS_PORT: 443
  TELNET_PORT: 23
  DNS_PORT: 53
  SNMP_PORT: 161
  POP3_PORT: 110
  IMAP_PORT: 143
  SMTP_PORT: 25
  MYSQL_PORT: 3306
  MSSQL_PORT: 1433
  NTP_PORT: 123
  OPENVPN_PORT: 1194
  PPTP_PORT: 1723
  FTP_PORT: 21
  RSYNC_PORT: 873
  SQUID_PORT: 3128
```

The `geoip-groups` relate to the `*-geoip.rules` sets. This allows you to set your organization's locations. The `*-geoip.rules` can then monitor for usage within your network from outside of your `HOME_COUNTRY`.

Example `geoip-groups` subsection:

```
# If you are using the -geoip rule sets & Sagan is compile with Maxmind
# GeoIP2 support (https://github.com/maxmind/libmaxminddb/releases),
# you'll want to define your $HOME_COUNTRY. ISO GeoIP country codes can
# be found at http://dev.maxmind.com/geoip/legacy/codes/iso3166/

geoip-groups:

   HOME_COUNTRY: "US,CA"
```

The aetas-groups relate to the *-aetas.rules sets. This allows you to define your organization's normal "work" hours. The *-aetas.rules can then monitor network usage and tool usage at defined hours of the day.

Example aetas-groups subsection:

```
# If you want to use -aetas, also known as time based rule sets,  you'll
# want to define the $SAGAN_HOURS and $SAGAN_DAYS variables. $SAGAN_HOURS is
# considered "normal" hours in a 24 hour clock format from "start time" to
# "end time".  $SAGAN_DAYS is the day of the week (0 == Sunday ->
# Saturday).  For more information,  see:

aetas-groups:

  SAGAN_HOURS: "0700-1800"
  SAGAN_DAYS: "12345"
```

The mmap-groups allow you to set variables used later in the sagan.yaml to set storage sizes for mmap() files. These variables are used later in the sagan-core section.

Example mmap-groups subsection:

```
# Variable for the max number of entries Sagan will retain via IPC.

mmap-groups:

  MMAP_DEFAULT: 10000
```

The misc-groups is a generic area to add variables. If you want to add a variable to the sagan.yaml file, this is likely the area you want to add them to.

Example misc-groups subsection:

```
misc-groups:

  CREDIT_CARD_PREFIXES: "4,34,37,300,301,302,303,304,305,2014,2149,309,36,38,39,54,55,
↪6011,6221,6222, 6223,6224,6225,6226,\
                         6227,6228,6229,644,645,646,647,648,649,65,636,637,638,639,22,
↪23,24,25,26,27,51,52,53,53,55"

   RFC1918: "10.,192.168.,172.16.,172.17.,172.18.,172.19.,172.20.,172.21.,172.22.,172.
↪23.,172.24.,172.25.,172.26.,172.27.,\
             172.28.,172.29.,172.30.,172.31."

  # $WINDOWS_DOMAINS is used by some Windows rule sets to determine if a log
  # message contains or does not contain a valid DOMAIN for your organization.
  # For more information, see:
  #
  # https://quadrantsec.com/about/blog/detecting_pass_the_hash_attacks_with_sagan_in_
↪real_time/

  WINDOWS_DOMAINS: "MYCOMPANYDOMAIN,EXAMPLEDOMAIN,ANOTHER_DOMAIN"

  # Known valid Microsoft PSExec MD5 sums.  Versions v1.98, v2.00, v2.10, v2.11, v2.
↪11 (2016).

  PSEXEC_MD5: "CD23B7C9E0EDEF184930BC8E0CA2264F0608BCB3,␣
↪9A46E577206D306D9D2B2AB2F72689E4F5F38FB1,\
               2EDEEFB431663F20A36A63C853108E083F4DA895,
↪B5C62D79EDA4F7E4B60A9CAA5736A3FDC2F1B27E,\
```

(continues on next page)

(continued from previous page)

```
                A7F7A0F74C8B48F1699858B3B6C11EDA"
```

CHAPTER 8

sagan-core

The sagan-core section defines internal Sagan core functionality. In this section, you can setup Sagan to receive data in different formats, how different data parsers work, tuning and other items.

The sagan-core is broken into subsections. They are core, parse_ip, selector, redis-server, mmap-ipc, ignore_list, geoip, liblognorm and plog.

8.1 core

The core subsection defines and sets some important information in the sagan.yaml configuration. Items like the default-host are used for when Sagan cannot normalize or find IP addresses it needs. The default default-port and default-proto are used for similar purposes.

One important item is the max-threads. This directly controls how much data Sagan can process at any given time. If you find yourself in a situation where Sagan is dropping logs, you likely need to increase this value.

The core is also the area where you can point Sagan to external data. For example, the classifications file assigns priority numbers to different classification levels. The references is a pointer to addresses that Sagan can point users to find more information about an alert.

The flexbit-storage tells Sagan "how" to store flexbit information. In most cases, you'll want to leave this default (mmap).

The input-type tells what format Sagan will receive data via the named PIPE (FIFO). Traditionally, Sagan uses a "pipe" delimited format. Sagan is increasingly moving to a JSON format and the JSON format will become the default. See the Syslog Configuration portion of this document for more information.

Example core subsection:

```
core:

  sensor-name: "default_sensor_name"  # Unique name for this sensor (no spaces)
  default-host: 192.168.2.1
  default-port: 514
```

(continues on next page)

(continued from previous page)

```
default-proto: udp
dns-warnings: disabled
source-lookup: disabled
fifo-size: 1048576          # System must support F_GETPIPE_SZ/F_SETPIPE_SZ.
max-threads: 100
classification: "$RULE_PATH/classification.config"
reference: "$RULE_PATH/reference.config"
gen-msg-map: "$RULE_PATH/gen-msg.map"
protocol-map: "$RULE_PATH/protocol.map"
flexbit-storage: mmap          # flexbit storage engine. ("mmap" or "redis")
xbit-storage: mmap             # xbit storage engine. ("mmap" or "redis")

# Sagan can sends logs in "batches" for performance reasons. In most
# environments, you'll likely want to set this to 10.  For more busy
# environments you may want to set this to 100.  This should allow Sagan
# to comfortably process up to 5k events per/second (EPS).  If you are
# looking at rates higher than 5k EPS,  please read:
#
# https://sagan.readthedocs.io/en/latest/high-performance.html
#
# The default setting is 1 which doesn't lead to the best performance.
# If you get more than 10 events per/second,  you might want to increase
# the batch-size to 10.

batch-size: 1

# Controls how data is read from the FIFO. The "pipe" setting is the traditional
# way Sagan reads in events and is the default. "json" is more flexible and
# will become the default in the future. If "pipe" is set, "json-map"
# and "json-software" have no function.

input-type: pipe                          # pipe or json
json-map: "$RULE_PATH/json-input.map"     # mapping file if input-type: json
json-software: syslog-ng                  # by "software" type.

# "parse-json-message" allows Sagan to detect and decode JSON within a
# syslog "message" field.  If a decoder/mapping is found,  then Sagan will
# extract the JSON values within the messages.  The "parse-json-program"
# tells Sagan to start looking for JSON within the "program" field.  Some
# systems (i.e. - Splunk) start JSON within the "program" field and
# into the "message" field.  This option tells Sagan to "append" the
# strings together (program+message) and then decode.  The "json-message-map"
# tells Sagan how to decode JSON values when they are encountered.

parse-json-message: disable
parse-json-program: disable
json-message-map: "$RULE_PATH/json-message.map"
```

8.1.1 sensor-name

The `sensor-name` is a unique human readable name of the Sagan instances. This is used to identify data sources. For example, Sagan can write `flexbits` to a shared database. The `sensor-name` can help identify which Sagan instance wrote which `flexbit`.

8.1.2 default-host

The `default-host` is the TCP/IP address of the Sagan system. This is used in cases where Sagan is unable to normalize data. Set this to your local IP addess.

8.1.3 default-port

The `default-port` is used when Sagan cannot normalize the destination port from a log message. When that happens, this value is used.

8.1.4 default-proto

The `default-proto` is the default protocol Sagan uses when the protocol cannot be normalized from a log message. Valid types are `udp`, ``tcp` and ``icmp``.

8.1.5 dns-warnings

If Sagan receives a hostname rather than an IP address from a syslog server, Sagan has the ability to do an "A record" lookup. If Sagan is unable to do a DNS lookup, it will emit a DNS warning message. The `dns-warnings` option disables those warnings. The `source-lookup` option must be enabled for this to have any effect. By default, this option is disabled.

8.1.6 source-lookup

If enabled, the `source-lookup` option will force Sagan to do a DNS A record lookup when it encounters a hostname rather than an IP address. Sagan performs some internal DNS caching but there is a performance penalty when this option is enabled. Also see `dns-warnings`. This option is disabled by default.

8.1.7 fifo-size

The `fifo-size` lets Sagan adjust the size of the named pipe (FIFO). The named pipe is how Sagan gets logs from syslog daemons like `rsyslog`, `syslog-ng` and `nxlog`. By default, most systems set the named pipe size at 63356 bytes. For performance reasons, we set the named pipe to the largest size possible. That size is 1048576 bytes, which is what Sagan defaults to. Valid values are 65536, 131072, 262144, 524288 and 1048576.

8.1.8 max-threads

The `max-threads` allows you to adjust how many worker threads Sagan spawns. Threads are what do the bulk of the log and data analysis work. Threads are used for CPU intensive analysis along with high latency operations. The busier the system is, the more threads you will need. Threads are also dependent on the type of `processors` enabled. Some `processors`, such as threat intelligence lookups require more time to complete. These require idle threads to do those lookups. The proper number of threads is largely dependent on several factors. Start at 100 and monitor the system's performance. While running Sagan in the foreground, monitor the `Thread Exhaustion` statistics. This will let you know if Sagan is running out of threads. If this number goes up, increase the number of threads available to Sagan. The default `max-threads` is set to 100.

8.1.9 classification

This points Sagan to the `classications.config`. The `classifications.config` is a file that maps classification types (ie - "attempted recon") to a priority level (ie - "1"). This data is used in rules via the `classtype` keyword.

https://github.com/beave/sagan-rules/blob/master/classification.config

8.1.10 gen-msg-map

The `gen-msg-map` is used to point `processors` to their "generator id". The Sagan engine uses an ID of "1". This file is used to assign other `processors` other IDs.

https://github.com/beave/sagan-rules/blob/master/gen-msg.map

8.1.11 reference

The `reference` option points Sagan to where the `reference.config` file is located on the file system. This file is used with the `reference` rule keyword.

https://github.com/beave/sagan-rules/blob/master/reference.config

8.1.12 protocol-map

The `protocol-map` is a simple method that Sagan can use to assign a TCP/IP protocol to a log message. The `protocol-map` contains either keywords to search for within a log "message" or within a "program" field. For example, if Sagan sees that the program "sshd" is in use, it will assign a TCP/IP protocol of TCP because the protocol SSH uses SSH. Another example might be a router log that contains the term "TCP" or "icmp" in it. Sagan will "see" this and assign the protocol within the log message internally. The `protocol-map` is used by the `parse_proto` rule keyword.

https://github.com/beave/sagan-rules/blob/master/protocol.map

8.1.13 flexbit-storage

The `flexbit-storage` tells Sagan how to store `flexbit` data. The default is `mmap` (memory mapped files). Sagan can also store flexbit data in a Redis database. To use the Redis value, Sagan will need to be compiled with `hiredis` support.

8.1.14 xbit-storage

The `xbit-storage` tells Sagan how to store `xbit` data. The default is `mmap` (memory mapped files). Sagan can also store xbit data in a Redis database. To use the Redis value, Sagan will need to be compiled with `hiredis` support.

8.1.15 batch-size

The `batch-size` option lets you set how much data can be passed from Sagan's master/main thread to "worker" threads (set by `max-threads`). This option can be very important in performance tuning in high data processing environments. The number specified in this option represents how many "log lines" will be passed. By default, it is set to 1. This means every time that Sagan gets a log line, it will pass it to a worker threads. This isn't very efficient and there is a performance penalty. If you are in an environment where you expect to process more than 10 events per/second (10 EPS), consider bumping this up to 10 or even the max of 100. If you are processing 50k EPS or more, see the "High Performance Considerations" of this document.

8.1.16 input-type

The `input-type` tells Sagan how to decode data it receives from the named pipe. There are two option; `pipe` or `json`. The `pipe` format is a legacy Sagan format. Data is received in the named pipe in a CSV format seperated by the '|' symbol. The newer `json` option tells Sagan to decode the data from the named pipe in a JSON format. When using the `json`, you will also need to set the `json-map` and `json-software`. If you are using the `pipe` value, no other options are needed. To use the `json` option, Sagan will need to be compiled with the `libfastjson` or `liblognorm`.

8.1.17 json-map

The `json-map` works in conjuction with the `input-type` of `json`. The `json-nap` tells Sagan where to load a mapping table of different software types (ie - `rsyslog`, `syslog-ng`, etc) and their associated JSON decode mappings. The data in this file is used with the `json-software` option to tell Sagan how do decode incoming JSON data from the named pipe. To use the `json-map` option, Sagan will need to be compiled with the `libfastjson` or `liblognorm`.

https://github.com/beave/sagan-rules/blob/master/json-input.map

8.1.18 json-sofware

The `json-software` tells Sagan which "map" to use from the `json-map` file that has been loaded. This mapping tells Sagan how to decode JSON data from the named pipe.

> To use the `json-software` option, Sagan will need to be compiled with the `libfastjson` or `liblognorm`.

8.1.19 parse-json-message:

The `parse-json-message` allows Sagan to automatically detect and decode JSON data within a "message" field of a log line. The option is used in conjuction with `parse-message-map` and requires that Sagan be compiled with `libfastjson` or `liblognorm` support.

8.1.20 parse-json-program:

The `parse-json-program` allows Sagan to detect JSON that starts within the "program" section of a log message. In certain situations, some systems start JSON within the "program" field rather than within the "message" field. When this happens, Sagan detects it and joins the "program" and "message" fields together (as one data source). Once that is done, the data can be decoded. This option is used in conjunction with `parse-message-map` and requires that Sagan be compiled with `libfastjson` or `liblognorm` support.

8.1.21 json-message-map:

The `json-message-map` logs a mapping table for use with `parse-json-message` and `parse-json-program`. When Sagan detects JSON via `parse-json-message` and/or via `parse-json-program`, it will attempt to apply mappings from this file. The "best mapping" wins. That is, the mapping with the most fields identified will "win" and Sagan will use that mapping with the log message. This can be useful for directly processing Suricata EVE logs and Splunk forwarded logs.

https://github.com/beave/sagan-rules/blob/master/json-input.map

8.2 parse_ip

The `parse_ip` subsection controls how the Sagan rule keywords `parse_src_ip` and `parse_dst_ip` function from within rules. The `ipv4-mapped-ipv6` determines how Sagan will work with IPv4 addresses mapped as IPv6. If `ipv4-mapped-ipv6` is enabled, Sagan will re-write IPv6 mapped addresses (for example ffff::192.168.1.1) to normal IPv4 notation (192.168.1.1).

Example `parse_ip` subsection:

```
# This controls how "parse_src_ip" and "parse_dst_ip" function within a rule.

parse-ip:
  ipv6: enabled                       # Parse IPv6 Addresses
  ipv4-mapped-ipv6: disabled          # Map ffff::192.168.1.1 back to 192.168.1.1
```

8.3 selector

The `selector` can be used in "multi-tenant" environments. This can be useful if you have multiple organizational logs going into one named pipe (FIFO) and you wish to apply rule logic on a per sensor/organization level. The `name` is the keyword that identifies the `selector`.

Example `selector` subsection:

```
# The "selector" adds "multi-tenancy" into Sagan.  Using the "selector" allows Sagan␣
↪to
# track IP source, IP destinations, etc. in order to ensure overlapping logs from␣
↪different
# environments are tracked separately.

selector:
  enabled: no
  name: "selector name"        # Log entry must be normalized and this value must
                               # be present in the normalized result
```

8.4 redis-server (experimental)

The `redis-server` is a beta feature that allows Sagan to store `flexbits` in a Redis database rather than a `mmap()` file. This can be useful in sharing `flexbits` across multiple platforms within a network. The `server` is the network address of your Redis server. The `port` is the network port address of the Redis server. The `password` is the Redis server's password. The `writer_threads` is how many Redis write threads Sagan should spawn to deal with Redis write operations.

Example `redis-server` subsection:

```
# Redis configuration.  Redis can be used to act as a global storage engine for
# flexbits.  This allows Sagan to "share" flexbit data across a network␣
↪infrastructure.
# This is experimental!

redis-server:

  enabled: no
  server: 127.0.0.1
  port: 6379
  #password: "mypassword"  # Comment out to disable authentication.
  writer_threads: 10
```

8.5 mmap-ipc

The `mmacp-ipc` subsection tells Sagan how much data to store in `mmap()` files and where to store it. The `ipc-directory` is where Sagan should store `mmap()` file. This is set to `/dev/shm` by default. On Linux systems `/dev/shm` is a ram drive. If you want to store `mmap()` files in a more permanent location, change the `ipc-directory`. Keep in mind, this may affect `mmap()` performance. The `flexbit`, `after`, `threshold` and `track-clients` are the max items that can be stored in `mmap()`. This typically defaults to 10,000 via the `$MMAP_DEFAULT` variable.

Example `mmap-ipc` subsection:

```
# Sagan creates "memory mapped" files to keep track of flexbits, thresholds,
# and afters.  This allows Sagan to "remember" threshold, flexbits and after
# data between system restarts (including system reboots!).

# This also allows Sagan to share information with other Sagan processes.
# For exampe, if one Sagan instance is monitoring "Linux" logs & another is
# monitoring "Windows" logs, Sagan can communicate between the two Sagan
# processes using these memory mapped files. A "flexbit" that is "set" by the
# "Linux" process is accessible and "known" to the Windows instance.

# The storage is pre-allocated when the memory mapped files are created
# The values can be increased/decreased by altering the $MMAP_DEFAULT
# variable. 10,000 entries is the system default.

# The default ipc-directory is /dev/shm (ram drive) for performance reasons.

mmap-ipc:

  ipc-directory: /dev/shm
  flexbit: $MMAP_DEFAULT
  after: $MMAP_DEFAULT
  threshold: $MMAP_DEFAULT
  track-clients: $MMAP_DEFAULT
```

8.6 ignore_list

The `ignore_list` subsection is a simple short circuit list of keywords. If Sagan encounters any keywords in this list, it is immediately dropped and not passed through the rest of the Sagan engine. In high throughput environments, this can save CPU time. The `ignore_file` is the location and file to load as an "ignore" list.

Example ignore_list subsection:

```
# A "short circuit" list of terms or strings to ignore.  If the the string
# is found in pre-processing a log message, it will be dropped.  This can
# be useful when you have log messages repeating without any useful
# information & you don't want to burn CPU cycles analyzing them.  Items
# that match will be "short circuit" in pre-processing before rules &
# processors are applied.

ignore_list:

  enabled: no
  ignore_file: "$RULE_PATH/sagan-ignore-list.txt"
```

8.6.1 geoip

The geoip subsection where you can configure Maxminds GeoIP settings. This includes enabling GeoIP lookups, where to find the Maxmind data files and what networks to "skip" GeoIP lookups. The country_database is the Maxmind database to load. The skip_networks option tells Sagan what networks not to lookup. The lookup_all_alerts forces Sagan to add GeoIP information to all alerts. When disabled, GeoIP information is only added to alerts when signatures with country_code is triggered.

Example geoip subsection:

```
# Maxmind GeoIP2 support allows Sagan to categorize events by their country
# code. For example, a rule can be created to track "authentication
# successes" & associate the country where the successful login came from.  If the
# successful login is from outside your country code,  via the $HOME_COUNTRY
# variable, an alert can be generated.  Sagan will need to be compiled with
# --enable-geoip2 flag.
#
# Maxmind GeoLite2 Free database:
# http://dev.maxmind.com/geoip/geoip2/geolite2/
#
# Country code (ISO3166):
# http://dev.maxmind.com/geoip/legacy/codes/iso3166/
#
# More information about Sagan & GeoIP, see:
# https://quadrantsec.com/about/blog/detecting_adversary_with_sagan_geoip/

geoip:

  enabled: no
  country_database: "/usr/local/share/GeoIP2/GeoLite2-Country.mmdb"
  lookup_all_alerts: true
  skip_networks: "8.8.8.8/32, 8.8.4.4/32"
```

8.7 liblognorm

liblognorm is a way that Sagan can extract useful information from a log file. For example, liblognorm is used to extract source and destination IP addresses, user names, MAC addresses, etc from log data. This option allows you to enable/disable the liblognorm functionality and where to load normalization rulebase files from (see normalize_rulebase). The normalize_rulebase is a mapping file that lets Sagan extract useful information from logs.

More information about `liblognorm` can be found in the *Prerequisites* section of the Sagan User Guide and the *LibLogNorm <https://FIXME>*_ web site.

Example `liblognorm` subsection:

```
# Liblognorm is a fast sample-based log normalization library.  Sagan uses
# this library to rapidly extract useful data (IP address, hashes, etc) from
# log messages.  While this library is not required it is recommended that
# Sagan be built with liblognorm enabled.  For more information, see:
#
# https://wiki.quadrantsec.com/bin/view/Main/LibLogNorm
#
# The normalize_rulebase are the samples to use to normalize log messages
# Sagan receives.

liblognorm:

  enabled: yes
  normalize_rulebase: "$RULE_PATH/normalization.rulebase"
```

8.8 plog

The `plog` functionality use to "sniff" syslog messages "off the wire". If you already have a centralized syslog server you are sending data, the data is not encrypted and is UDP, this option can be used to "sniff" logs while they are in transit to your centralized logging system. In order to "sniff" the logs, you will need a "span" port or "tap". This option can be useful when testing Sagan's functionality. This should not be used in production environments since the robustness of "sniffing" varies. The `interface` option is the network device you want to "sniff" traffic on. the `bpf` (Berkely Packet Filter) is the filter to use to extract logs from the network. The `log-device` is where Sagan will inject logs after they are "sniffed" off the network. The `promiscuous` option puts the network interface Sagan is using in "promiscious mode" or not.

Example `plog` subsection:

```
# 'plog',  the promiscuous syslog injector, allows Sagan to 'listen' on a
# network interface and 'suck' UDP syslog messages off the wire.  When a
# syslog packet is detected, it is injected into /dev/log.  This is based
# on work by Marcus J. Ranum in 2004 with his permission.
#
# For more information,  please see:
#
# https://raw.githubusercontent.com/beave/sagan/master/src/sagan-plog.c

plog:

  enabled: no
  interface: eth0
  bpf-filter: "port 514"
  log-device: /dev/log     # Where to inject sniffed logs.
  promiscuous: yes
```

CHAPTER 9

processors

Sagan `processors` are methods of detection outside of the Sagan rule engine.

9.1 track-clients

The `track-clients` processor is used to detect when a syslog client has stopped or restarted sending logs to Sagan. This can be useful for detecting systems where logging has been disabled. In the event a syslog client stops sending logs, Sagan generates an alert for notification purposes. When the syslog client comes back online, Sagan will generate another alert for notification purposes. The `time` is how long a syslog client has not sent a log message to be considered "down".

Example `track-clients` subsection:

```
# The "tracking clients" processor keeps track of the systems (IP addresses),
# reporting to Sagan.  If Sagan stops receiving logs from a client for a
# specified amount of time ("timeout"), an alert/notification is created.
# When the system comes back online,  another alert/notification is
# created.

- track-clients:
    enabled: no
    timeout: 1440             # In minutes
```

9.2 rule-tracking

The `rule-tracking` processor is used to detect unused rule sets. This can be useful for detecting when rules are loaded which do not need to be. Rules that are loaded that are not used waste CPU cycles. This assists with rule tuning. The `console` option allows for rule tracking statistics to the console when Sagan is being run in the foreground. The `syslog` option tells Sagan to send rule tracking statistics to syslog. The `time` option tells Sagan how often to record rule tracking statistics (in minutes).

Example rule-tracking subsection:

```
# This reports on rule sets that have and have not "fired".  This can be
# useful in tuning Sagan.

- rule-tracking:
    enabled: yes
    console: disabled
    syslog: enabled
    time: 1440                # In minutes
```

9.3 perfmonitor

** PERFMON has been deperciated for JSON stats as of 2.0.1 **

The perfmonitor processor records Sagan statistics to a CSV file. This can provide useful data about detection and the performance of Sagan. The time option sets how often Sagan should record perfmonitor data.

Example perfmonitor subsection:

```
# The "perfmonitor" processor writes statistical information every specified
# number of seconds ("time") to a CSV file.  This data can be useful for
# tracking the performance of Sagan.  This data can also be used with
# RRDTool to generate graphs.

- perfmonitor:
    enabled: no
    time: 600
    filename: "$LOG_PATH/stats/sagan.stats"
```

9.4 blacklist

The blacklist processor reads in a file at load time (or reload) that contains IP addresses you wish to alert on. Detection is controlled by the *-blacklist.rules rule sets. The idea is to load IP addresses of interest into this list and Sagan can monitor for them. The list is a file containing IP and network addresses in a CIDR format (ie - 192.168.1.0/24, 10.0.0.0/8).

Example perfmonitor subsection:

```
# The "blacklist" process reads in a list of hosts/networks that are
# considered "bad".  For example, you might pull down a list like SANS
# DShield (http://feeds.dshield.org/block.txt) for Sagan to use.  If Sagan
# identifies any hosts/networks in a log message from the list, an alert
# will be generated.  The list can be in a IP (192.168.1.1) or CIDR format
# (192.168.1.0/24).  Rules identified as -blacklist.rules use this data.
# You can load multiple blacklists by separating them with commas.  For
# example; filename: "$RULE_PATH/list1.txt, $RULE_PATH/list2.txt".

- blacklist:
    enabled: no
    filename: "$RULE_PATH/blacklist.txt"
```

9.5 bluedot

The `bluedot` processor looks up data in the Quadrant Information Security "Bluedot" Threat Intelligence database. This is done over a `http` session. Access to this database is not public at this time.

Example `bluedot` subsection:

```
# The "bluedot" processor extracts information from logs (URLs, file hashes,
# IP address) and queries the Quadrant Information Security "Bluedot" threat
# intelligence database.  This database is 'closed' at this time.  For more
# information,  please contact Quadrant Information Security @ 1-800-538-9357
# (+1-904-296-9100) or e-mail info@quadrantsec.com for more information.
# Rules identified with the -bluedot.rules extension use this data.

- bluedot:
    enabled: no
    device-id: "Device_ID"
    cache-timeout: 120
    categories: "$RULE_PATH/bluedot-categories.conf"

    max-ip-cache: 300000
    max-hash-cache: 10000
    max-url-cache: 20000
    max-filename-cache: 1000

    ip-queue: 1000
    hash-queue: 100
    url-queue: 1000
    filename-queue: 1000

    host: "bluedot.qis.io"
    ttl: 86400
    uri: "q.php?qipapikey=APIKEYHERE"

    skip_networks: "8.8.8.8/32, 8.8.4.4/32"
```

9.6 zeek-intel (formally "bro-intel")

The `zeek-intel` (formally known as `bro-intel`) allows Sagan to load files from the "Zeek (Bro) intelligence framwork". This allows Sagan to lookup IP address, hashes and other data from Zeek Intelligence data.

Example `zeek-intel` subsection:

```
# The "zeek-intel" (formally "bro-intel") processor allows Sagan to use
# threat intelligence data from the "Zeek (Bro) Intelligence Framework".
# Rules identified with the # -brointel.rules use this data.  For more information
# about this processor,  see:
#
# https://quadrantsec.com/about/blog/using_sagan_with_bro_intelligence_feeds/
# https://wiki.quadrantsec.com/bin/view/Main/SaganRuleReference#bro_intel_src_ipaddr_
↪dst_ipaddr
# http://blog.bro.org/2014/01/intelligence-data-and-bro_4980.html
# https://www.bro.org/sphinx-git/frameworks/intel.html
#
# A good aggregate source of Bro Intelligence data is at:
```

(continues on next page)

(continued from previous page)

```
#
# https://intel.criticalstack.com/

- zeek-intel:
    enabled: no
    filename: "/opt/critical-stack/frameworks/intel/master-public.bro.dat"
```

9.7 dynamic-load

The `dynamic-load` processor will detect new logs entering the Sagan engine and can either automatically load rules or send an alert about new logs being detected. The idea here is to have Sagan assist with the detection of network and hardware changes. This rule is tied to the `dynamic.rules` rule set. The `dynamic.rules` rule set has signatures used to detect new log data entering the Sagan engine. The `sample-date` controls how often to look for new logs entering the Sagan engine. The higher the `sample-rate` the less CPU is used but the longer it will take to detect new data. The lower the `sample-rate` the faster Sagan can detect new data but at a higher cost to the CPU. The `type` can be `dynamic_load` or `log_only`. If set to `dynamic_load`, when new data is detected, Sagan will automatically load the associated rule from the `dynamic.rules`. If set to `log_only`, Sagan will not load any data and only generate an alert that new data was detected.

Example `dynamic-load` subsection:

```
# The 'dynamic_load' processor uses rules with the "dynamic_load" rule option
# enabled. These rules tell Sagan to load additional rules when new log
# traffic is detected.  For example,  if Sagan does not have 'proftpd.rules'
# enabled but detects 'proftp' log traffic,  a dynamic rule can automatically
# load the 'proftpd.rules' for you.  Dynamic detection rules are named
# 'dynamic.rules' in the Sagan rule set.  The "sample-rate" limits amount of
# CPU to dedicated to detection new logs. The "type" informs the process
# "what" to do.  Valid types are "dynamic_load" (load & alert when new rules
#  are loaded), "log_only" (only writes detection to the sagan.log file) and
# "alert" (creates an alert about new logs being detected).

- dynamic-load:
    enabled: no
    sample-rate: 100          # How often to test for new samples.
    type: dynamic_load        # What to do on detection of new logs.
```

CHAPTER 10

outputs

Sagan supports writing data in various formats. Some formats may be more suitable for humans to read, while others might be better for outputing to databases like Elasticsearch and MySQL.

10.1 eve-log

Sagan can write to Suricata's "Extensible Event Format", better known as "EVE". This is a JSON format in which events (alerts, etc) are written to. This data can then be used to transport data into Elasticsearch (using software like Logstash) or Meer (for MySQL/MariaDB/PostgreSQL) output. If you are looking to get alert data into any database back end, you'll likely want to enable this output plugin.

Example `eve-log` subsection:

```
outputs:

  # EVE alerts can be loaded into software like Elasticsearch and is a good
  # replacement for "unified2" with software like "Meer".  For more
  # information on Meer, Check out:
  #
  # https://github.com/beave/meer

  - eve-log:
      enabled: no
      interface: logs
      alerts: yes                     # Logs alerts
      logs: no                        # Send all logs to EVE.
      filename: "$LOG_PATH/eve.json"
```

10.2 alert

The `alert` format is a simple, multiline human readable format. The output is similar to that of traditional `Snort` alert log.

Example `alert` subsection:

```
# The 'alert' output format allows Sagan to write alerts, in detail, in a
# traditional Snort style "alert log" ASCII format.

- alert:
    enabled: yes
    filename: "$LOG_PATH/alert.log"
```

10.3 fast

The `fast` format is a simple, single line human readable format. The output is similar to the traditional `Snort` "fast" log.

Example `fast` subsection:

```
# The 'fast' output format allows Sagan to write alerts in a format similar
# to Snort's 'fast' output format.

- fast:
    enabled: no
    filename: "$LOG_PATH/fast.log"
```

10.4 smtp

The `smtp` output allows Sagan to send alerts via e-mail.

Example `smtp` subsection:

```
# The 'smtp' output allows Sagan to e-mail alerts that trigger.  The rules
# you want e-mailed need to contain the 'email' rule option and Sagan must
# be compiled with libesmtp support.

- smtp:
    enabled: no
    from: sagan-alert@example.com
    server: 192.168.0.1:25
    subject: "** Sagan Alert **"
```

10.5 syslog

The `syslog` output plugin writes alerts to the system's syslog that Sagan is running on. This can be useful for forwarding Sagan alert data to other SIEMs.

Example `syslog` subsection:

```
# The 'syslog' output allows Sagan to send alerts to syslog. The syslog
# output format used is exactly the same as Snort's.  This means that your
# SIEMs Snort log parsers should work with Sagan.

- syslog:
    enabled: no
    facility: LOG_AUTH
    priority: LOG_ALERT
    extra: LOG_PID
```

CHAPTER 11

rule-files

The `rule-files` section tells Sagan what "rules" to load. This can be a list of files or rules that can be broken out into seperate `include`.

Example `rule-files` subsection:

```
rules-files:

  ##############################################################################
  # Dynamic rules - Only use if you have the 'dynamic_load' processor enabled #
  ##############################################################################

  #- $RULE_PATH/dynamic.rules

  ##############################################################################
  # GeoIP rules - Only use if you have $HOME_COUNTRY and 'geoip' core enabled #
  ##############################################################################

  #- $RULE_PATH/cisco-geoip.rules
  #- $RULE_PATH/citrix-geoip.rules
  #- $RULE_PATH/courier-geoip.rules
  #- $RULE_PATH/f5-big-ip-geoip.rules
  #- $RULE_PATH/fatpipe-geoip.rules
  #- $RULE_PATH/fortinet-geoip.rules
  #- $RULE_PATH/imapd-geoip.rules
  #- $RULE_PATH/juniper-geoip.rules
  #- $RULE_PATH/openssh-geoip.rules
  #- $RULE_PATH/proftpd-geoip.rules
  #- $RULE_PATH/riverbed-geoip.rules
  #- $RULE_PATH/snort-geoip.rules
  #- $RULE_PATH/ssh-tectia-server-geoip.rules
  #- $RULE_PATH/vmware-geoip.rules
  #- $RULE_PATH/vsftpd-geoip.rules
  #- $RULE_PATH/windows-geoip.rules
  #- $RULE_PATH/windows-owa-geoip.rules
  #- $RULE_PATH/zimbra-geoip.rules
```

(continues on next page)

(continued from previous page)

```
##############################################################################
# Aetas rules - Only use if $SAGAN_HOUR/$SAGAN_DAY is defined!              #
##############################################################################

#- $RULE_PATH/cisco-aetas.rules
#- $RULE_PATH/fatpipe-aetas.rules
#- $RULE_PATH/fortinet-aetas.rules
#- $RULE_PATH/juniper-aetas.rules
#- $RULE_PATH/openssh-aetas.rules
#- $RULE_PATH/proftpd-aetas.rules
#- $RULE_PATH/riverbed-aetas.rules
#- $RULE_PATH/ssh-tectia-server-aetas.rules
#- $RULE_PATH/windows-aetas.rules

##############################################################################
# Malware rules - Rules useful for detecting malware.                        #
##############################################################################

#- $RULE_PATH/cisco-malware.rules
#- $RULE_PATH/fortinet-malware.rules
#- $RULE_PATH/nfcapd-malware.rules
#- $RULE_PATH/proxy-malware.rules
#- $RULE_PATH/windows-malware.rules

##############################################################################
# Bro Intel rules - Make sure the 'bro-intel processor is enabled!          #
##############################################################################

#- $RULE_PATH/cisco-brointel.rules
#- $RULE_PATH/citrix-brointel.rules
#- $RULE_PATH/windows-brointel.rules
#- $RULE_PATH/windows-owa-brointel.rules
#- $RULE_PATH/bro-intel.rules

##############################################################################
# Bluedot rules - Make sure the 'bluedot' processor is enabled!             #
##############################################################################

#- $RULE_PATH/bluedot.rules
#- $RULE_PATH/bro-bluedot.rules
#- $RULE_PATH/cisco-bluedot.rules
#- $RULE_PATH/citrix-bluedot.rules
#- $RULE_PATH/courier-bluedot.rules
#- $RULE_PATH/f5-big-ip-bluedot.rules
#- $RULE_PATH/fatpipe-bluedot.rules
#- $RULE_PATH/fortinet-bluedot.rules
#- $RULE_PATH/imapd-bluedot.rules
#- $RULE_PATH/juniper-bluedot.rules
#- $RULE_PATH/openssh-bluedot.rules
#- $RULE_PATH/proftpd-bluedot.rules
#- $RULE_PATH/riverbed-bluedot.rules
#- $RULE_PATH/snort-bluedot.rules
#- $RULE_PATH/ssh-tectia-server-bluedot.rules
#- $RULE_PATH/vmware-bluedot.rules
#- $RULE_PATH/vsftpd-bluedot.rules
#- $RULE_PATH/windows-bluedot.rules
```

(continues on next page)

(continued from previous page)

```
#- $RULE_PATH/windows-owa-bluedot.rules

################################################################################
# Correlated rules - Rules that use xbits/flexbit to detect malicious behavior #
################################################################################

- $RULE_PATH/cisco-correlated.rules
- $RULE_PATH/citrix-correlated.rules
- $RULE_PATH/courier-correlated.rules
- $RULE_PATH/fatpipe-correlated.rules
- $RULE_PATH/fortinet-correlated.rules
- $RULE_PATH/imapd-correlated.rules
- $RULE_PATH/openssh-correlated.rules
- $RULE_PATH/ssh-tectia-server-correlated.rules
- $RULE_PATH/vmware-correlated.rules
- $RULE_PATH/vsftpd-correlated.rules
- $RULE_PATH/windows-correlated.rules
- $RULE_PATH/windows-owa-correlated.rules

##############################################################################
# Standard rules - Rules that do not require any dependencies.              #
##############################################################################

#- $RULE_PATH/as400.rules
- $RULE_PATH/adtran.rules
- $RULE_PATH/apache.rules
- $RULE_PATH/apc-emu.rules
- $RULE_PATH/arp.rules
#- $RULE_PATH/artillery.rules
- $RULE_PATH/asterisk.rules
- $RULE_PATH/attack.rules
- $RULE_PATH/barracuda.rules
- $RULE_PATH/bash.rules
- $RULE_PATH/bind.rules
- $RULE_PATH/carbonblack.rules
- $RULE_PATH/bonding.rules
- $RULE_PATH/bro-ids.rules
- $RULE_PATH/cacti-thold.rules
#- $RULE_PATH/cisco-acs.rules
- $RULE_PATH/cisco-ise.rules
- $RULE_PATH/cisco-cucm.rules
- $RULE_PATH/cisco-ios.rules
- $RULE_PATH/cisco-meraki.rules
- $RULE_PATH/cisco-pixasa.rules
#- $RULE_PATH/cisco-prime.rules
- $RULE_PATH/cisco-wlc.rules
- $RULE_PATH/citrix.rules
- $RULE_PATH/courier.rules
- $RULE_PATH/cylance.rules
#- $RULE_PATH/deleted.rules
#- $RULE_PATH/digitalpersona.rules
- $RULE_PATH/dovecot.rules
- $RULE_PATH/f5-big-ip.rules
- $RULE_PATH/fatpipe.rules
- $RULE_PATH/fipaypin.rules
- $RULE_PATH/fortinet.rules
- $RULE_PATH/ftpd.rules
```

(continues on next page)

(continued from previous page)

```
  - $RULE_PATH/grsec.rules
  - $RULE_PATH/honeyd.rules
  #- $RULE_PATH/hordeimp.rules
  #- $RULE_PATH/hostapd.rules
  - $RULE_PATH/huawei.rules
  - $RULE_PATH/imapd.rules
  - $RULE_PATH/ipop3d.rules
  - $RULE_PATH/juniper.rules
  #- $RULE_PATH/kismet.rules
  - $RULE_PATH/knockd.rules
  - $RULE_PATH/linux-kernel.rules
  - $RULE_PATH/milter.rules
  - $RULE_PATH/mongodb.rules
  - $RULE_PATH/mysql.rules
  - $RULE_PATH/nexpose.rules
  - $RULE_PATH/nfcapd.rules
  - $RULE_PATH/nginx.rules
  - $RULE_PATH/ntp.rules
  - $RULE_PATH/openssh.rules
  - $RULE_PATH/openvpn.rules
  - $RULE_PATH/oracle.rules
  - $RULE_PATH/palo-alto.rules
  - $RULE_PATH/php.rules
  - $RULE_PATH/postfix.rules
  - $RULE_PATH/postgresql.rules
  - $RULE_PATH/pptp.rules
  - $RULE_PATH/procurve.rules
  - $RULE_PATH/proftpd.rules
  - $RULE_PATH/pure-ftpd.rules
  - $RULE_PATH/racoon.rules
  - $RULE_PATH/riverbed.rules
  - $RULE_PATH/roundcube.rules
  - $RULE_PATH/rsync.rules
  - $RULE_PATH/samba.rules
  - $RULE_PATH/sendmail.rules
  - $RULE_PATH/snort.rules
  - $RULE_PATH/solaris.rules
  - $RULE_PATH/sonicwall.rules
  - $RULE_PATH/squid.rules
  - $RULE_PATH/ssh-tectia-server.rules
  - $RULE_PATH/su.rules
  - $RULE_PATH/symantec-ems.rules
  - $RULE_PATH/syslog.rules
  - $RULE_PATH/tcp.rules
  - $RULE_PATH/telnet.rules
  - $RULE_PATH/trendmicro.rules
  - $RULE_PATH/tripwire.rules
  - $RULE_PATH/vmpop3d.rules
  - $RULE_PATH/vmware.rules
  - $RULE_PATH/vpopmail.rules
  - $RULE_PATH/vsftpd.rules
  - $RULE_PATH/web-attack.rules
  #- $RULE_PATH/weblabrinth.rules
  - $RULE_PATH/windows-applocker.rules
  - $RULE_PATH/windows-auth.rules
  - $RULE_PATH/windows-emet.rules
  - $RULE_PATH/windows-misc.rules
```

(continues on next page)

(continued from previous page)

```
  - $RULE_PATH/windows-mssql.rules
  - $RULE_PATH/windows-security.rules
  - $RULE_PATH/windows-owa.rules
  - $RULE_PATH/windows.rules
  - $RULE_PATH/windows-sysmon.rules
  - $RULE_PATH/wordpress.rules
  - $RULE_PATH/xinetd.rules
  - $RULE_PATH/yubikey.rules
  - $RULE_PATH/zeus.rules
  - $RULE_PATH/zimbra.rules

#
# Include other configs
#

# Includes.  Files included here will be handled as if they were
# included in this configuration file.

#include: "/usr/local/etc/include1.yaml"
#include: "$RULE_PATH/include2.yaml"
```

CHAPTER 12

Rule syntax

Sagan rule syntax is very similar to that of Suricata or Snort . This is was intentionally done to maintain compatibility with rule management software like `oinkmaster` and `pulledpork` and allows Sagan to correlate log events with your Snort/Suricata IDS/IPS system.

This also means that if you are already familiar with signature writing in Suricata and Snort, you already understand the Sagan syntax!

To understand the basic Sagan rule syntax, we will be using the following simple rule. This section of the Sagan user guide only covers up to the first *rule option*. That is, this section will cover up to the `msg` portion of this rule only. The rest of the rule is considered `rule options`.

Basic Sagan rule:

```
alert any $EXTERNAL_NET any -> $HOME_NET any (msg: "[SYSLOG] System out of disk space
↪"; pcre: "/file system full|No space left on device/i"; classtype: hardware-event;␣
↪threshold: type limit, track by_src, count 1, seconds 300; reference: url,wiki.
↪quadrantsec.com/bin/view/Main/5000116; sid:5000116; rev:2;)
```

`alert`

This informs Sagan how to flag the event. Valid options are `alert`, `drop` or `pass`. When using the `pass` option and the signatures conditions are met, no other signatures are processed.

`any`

Valid options for this field are `any`, `tcp`, `udp` or `icmp`. In most cases, you will likely want to specify `any`. The protocal is determined by the `parse_proto`, `parse_proto_program` or liblognorm rule keywords.

`$EXTERNAL_NET`

This informs Sagan where the source IP address or addresses must be coming from in order to trigger. By default the variable `$EXTERAL_NET` is used. This is set in the `sagan.yaml` configurations file and defaults to `any`. most cases, "any" (any source) is what you want. In other cases, you might want the signature to trigger when it is from a particular host. For example:

192.168.1.1

Makes Sagan only trigger if the source of the event is from the address 192.168.1.1 (/32 is automatically assumed). You can also apply multiple networks. For example:

[192.168.1.0/24, 10.0.0.0/24]

Is valid and will only trigger if the network address is within 192.168.1.0/24 or 10.0.0.0/24. You can also apply *not* logic to the addresses. For example.

!192.168.1.1/32

This will only trigger when the IP address is *not* 192.168.1.1.

This filed is populated by whatever the source IP address within the log might be. For example, if the signature lacks `parse_src_ip` or `normalize` (see rule options), then the syslog source is adopted. If `parse_src_ip` or `normalize` rule option is used, then data (if any) that is extracted from the log is used.

`any`

The next `any` is the source port. If the `normalize` or `default_src_port` rule option is used, it will be applied here. This can be useful in filtering out certain subnets or syslog clients.

`->`

This would be the direction. From the $EXTERNAL_NET -> $HOME_NETWORK.

`$HOME_NETWORK`

This works similarly to how $EXTERNAL_NET functions. Rather than being the source of the traffic, this is the destination of the traffic. Like $EXTERNAL_NET, this is set in the `sagan.yaml` configuration file and defaults to `any`. Also like the $EXTERNAL_NET, network CIDR notation can be used (ie - 192.168.1.0). Data from this is populated by the `parse_dst_ip` and `normalize` rule options.

`any`

The final rule option is the destination port. If the `normalize` or `default_dst_port` rule option is used, it will be applied here. This can be useful in filtering out events from certain subnets.

CHAPTER 13

Rule Keywords

13.1 after

after: track {by_src|by_dst|by_username|by_string}, **count** {number of event}, **seconds** {numbe

"after" is used to trigger an alert "after" a number of events have happened within a specific amount of time. "after" tracks by the source or destination IP address of the event. The example would track events by the source IP address. If the event is triggered more than 10 times within 300 seconds (5 minutes), an alert is triggered.

after: track by_src, count 10, seconds 300;

After can be tracked by multiple 'track' options. For example:

after: track by_src&by_username, count 5, seconds 300;

The above would track by the source IP address and by the username.

13.2 alert_time

alert_time: days {days}, **hours** {hours};

"alert_time" allows a rule to only trigger on certain days and/or certain hours. For example, let's assume that Windows RDP (Remote Desktop Protocol) is normal between the hours of 0800 (8 AM) to 1800 (6 PM). However, RDP sessions outside of that time-frame would be considered suspicious. This allows you to build a rule that will trigger outside of the "normal RDP" times.

Days are represented via digits 0 - 6. 0 = Sunday, 1 Monday, 2 Tuesday, 3 Wednesday, 4 Thursday, 5 Friday, 6 = Saturday.

Hours are represented by the 24 hour clock.

alert_time: days 0123456, hours 0800-1800;

The example above would cause a rule to trigger every day of the week between the hours of 0800 (8:00 AM) to 1800 (6:00 PM). One caveat is with "between" days. For example, if you wanted to create an alert_time rule that stretches from Monday 2300 (11 PM) to Tuesday 0700 (7 AM). The format would be:

alert_time: days 1, hours 2300-0800;

You do not need to include Tuesday (2) in the "days" option. Since the times stretch between two days, Sagan will automatically take this into consideration and make the adjustments. If you were to include "days 12", this would cause Sagan to alert on Monday-Tuesday between 2300 - 0800 and Tuesday-Wednesday 2300-0800.

alert_time can also be used with sagan.yaml variables. For example, if you have "SAGAN_DAYS: 12345" and "SAGAN_HOURS: 0800-1300" in your sagan.yaml (see "aetas-groups" in your sagan.yaml), you could then create a rule like this:

alert_time: days $SAGAN_DAYS, hours $SAGAN_HOURS;

13.3 append_program

`append_program;`

The `append_program` rule option forces the syslog program field to be appended to the syslog message. This can be useful when the program fields are unpredictable. An example of this are Cisco ASAs with 'Emblem' enabled or disabled. When Cisco "Emblem" is disabled, the syslog "program" field will contain the Cisco ASA 'status' code (i.e - '%ASA-3-114006'). However, when Cisco Emblem is enabled, the syslog "program" field gets shifted up in order and becomes part of the syslog "message". The result is signatures that do not fire despite the status code being present.

The `append_program` option will append the program field to the end of the syslog message. This way, the rule writer can use rule options like `content`, `pcre`, etc to detect the status code regardless of if the syslog program has been shifted or not.

Sagan will append the syslog program as "syslog message | syslog program".

13.4 blacklist

`blacklist` `{by_src|by_dst|both|all};`

This looks up the TCP/IP address that was parsed via `normalize`, `parse_src_ip` or `parse_dst_ip` from a "blacklist" file. The "blacklist" file is a file that contains IPv4 and IPv6 addresses in CIDR notation from that file. In order to use this option the `sagan.yaml` processors `blacklist` must be enabled.

blacklist: by_src; parse_src_ip: 1;

13.5 bluedot

`bluedot:` `type {ip_reputation},track {src|dst|both|all},{none|mdate_effective_period|cdate_e`

`bluedot:` `type {file_hash|url|filename},{category};`

Bluedot is Quadrant Information Security's Threat Intelligence database that Sagan can query. In order to use this functionality you will need a Quadrant Information Security API key and have the `bluedot` processors enabled.

As Sagan extracts data like IP addresses, file hashes, URLs and filenames, Sagan can query the Bluedot database to determine if they are hostile or not. These types of lookups can be incorporated into signatures. For example:

bluedot: type ip_reputation, track by_src, none, Malicious,Tor,Honeypot,Proxy;

This will lookup the source IP out of the Bluedot database for *Malicious*, *Tor*, *Honeypot* or *Proxy* activity. If the source IP address is found in any of these categories, the option will fire.

In some cases, you might not want to trigger on older IoCs. To filter out older data from Bluedot you can use the `mdate_effective_period` (last modification of the IoC) or `cdate_effective_period` (creation date of the IoC). For example:

bluedot: type ip_reputation, track all, mdate_effective_period 1 months, Malicious,Tor,Proxy;

This will query all TCP/IP addresses found in a log line and query for *Malicious*, *Tor* and *Proxy* addresses that are no older than one month old. If the time is set to `none`, then any IoCs found for a TCP/IP address are returned regardless of `mdate_effective_period` or `cdate_effective_period`.

Below is an example of querying a file hash in Bluedot

bluedot: type file_hash,Malicious; parse_hash: sha1;

13.6 classtype

`classtype:` `{classification}`

This links the rule to a classification. Classification can be used to determine priority level. For example:

classtype: exploit-attempt;

A "exploit-attempt" classification is a priority 1 (highest) level event. For a complete list of classification types, see http://github.com/beave/sagan-rules/blob/master/classification.config

13.7 content

content is a simple means of determining if the {search} string is in an event/syslog message. For example:

content: "authentication failure";

Will search a log message for the term "authentication failure". content can also be used as part of a NOT statement. For example:

content:!"frank";

This means that the message does NOT contain the term "frank". Tied together, we can make statements like:

content: "authentication failure"; content:!"frank";

If the term "authentication failure" is found and does NOT contain the term "frank", then the rule will trigger. Otherwise, the event is ignored.

content: "User Agent|3a| Testing";

This tells content to search for "User Agent: Testing". The **|3a|** is a hex encoded option for a ":". You can use multiple hex encoded options. For example, "**|3a 3b 3c|**". Hex values can also be broken up. For example, "This **|3a|** is a testing with **|3b|** in it".

13.8 country_code

`country_code:` `track {by_src|by_dst},` **`{is|isnot}`** `{ISO3166 Country Codes}`

Used to track events from specific countries.

country_code: track by_src, isnot US;

The example above means, "track by the source address of the event. If the GeoIP 2 location is not from the United States, trigger the rule".

country_code: track by_dst, is [CN,RU,HK];

The example above means, "track by the destination address of the event. If the GeoIP 2 location is going to China, Russia or Hong Kong, trigger the rule".

Country codes are based on ISO3166. See http://dev.maxmind.com/geoip/legacy/codes/iso3166/ for the full listing.

Typically, country codes are tied to the sagan.yaml variable $HOME_COUNTRY (See "geoip-groups" in the sagan.yaml). For example:

country_code: track by_src, isnot $HOME_COUNTRY;

Note: This requires GeoIP2 support to be compiled into Sagan

13.9 default_proto

`default_proto:` `{tcp/udp/icmp}`

The default_proto sets the default protocol in the event normalization fails. For example, OpenSSH uses the TCP protocol. However, OpenSSH log messages do not specify the protocol in use. By using the rule option default_proto, Sagan will assign the protocol specified by the rule writer when triggered. This option can be overridden by parse_proto or liblognorm (if used).

Valid values are icmp, tcp and udp or defined variables (ie - "$PROTOCOL"). Defaults to the Sagan YAML "default-proto".

13.10 default_dst_port

`default_dst_port:` `{port number}`

The default_dst_port sets the default port number in the event normalization fails. For example, OpenSSH typically uses port 22. However, OpenSSH log messages do not specify the port being used. By using the rule option default_dst_port, Sagan will assign the port specified by the rule writer when triggered. This option can be overridden by liblognorm.

Valid values are integers (1-63556) or defined variables (ie - "$SSH_PORT"). Defaults to the Sagan YAML "default-port".

13.11 default_src_port

`default_src_port:` `{port number}`

The default_src_port sets the default port number in the event normalization fails. For example, if a log message does not contain the source port, this value is used instead. This can be overridden by liblognorm.

Valid values are integers (1-63556) or defined variables (ie - "$SOURCE_PORT). Defaults to the Sagan YAML "default-port".

Note: This requires GeoIP support to be compiled into Sagan

13.12 depth

depth: `{depth value}`

The depth keyword allows the rule writer to specify how far into a log line Sagan should search for the specified pattern from a given offset.

For example:

content: "bob"; depth: 10;

This would start searching at the beginning of the log line (default offset: 0) and search only 10 bytes deep for the term "bob".

Example with offset and depth used together:

content: "bob"; offset: 5; depth: 10;

Sagan will start searching for the term "bob" when it gets to 5 bytes into the log line (see offset). It will only search for "bob" after the offset for 10 bytes.

This function is identical to Snort's "depth" rule option. For more information see: http://blog.joelesler.net/2010/03/offset-depth-distance-and-within.html

13.13 distance

distance: `{distance value}`

The distance keyword allows the rule writer to specify how far into a log line Sagan should ignore before starting to search for the specified pattern relative to the end of the previous pattern match.

For example:

content:"GET"; depth:3; content:"downloads"; distance:10;

This will cause Sagan to look for the word "GET" within the first 3 bytes (depth) of the log line. The next content will start looking for the term "downloads" 10 bytes away from the previous depth. The above would match on the term "GET /content/downloads" but not "GET /download". The " /content/" (10 bytes) is skipped over in the distance.

This function is identical to Snort's "distance" rule option. For more information see: http://blog.joelesler.net/2010/03/offset-depth-distance-and-within.html

13.14 dynamic_load

{dynamic_load: `/path/to/rules/to/load}`

This option works in conjunction with the `sagan.yaml` `dynamic_load` configuration. When a rule is triggered with this option enabled, Sagan will dynamically load the rules. This is useful for detecting new logs introduced to the system where rules are not enabled. For more information, see https://quadrantsec.com/about/blog/dynamic_rules_with_sagan/

dynamic_load: $RULE_PATH/oracle.rules;

13.15 email

email: `{email address}`

If present in a rule, Sagan will e-mail the event to the email address supplied.

email: bob@example.org;

Note: This requires Sagan to be compiled with libesmtp support.

13.16 event_id

`event_id``: {id},{id},{id}...;`

This option attempts to locate an "Event ID" in a syslog message or within JSON data. This is typically used with Microsoft Windows event IDs but is not limited to this. When searching log data, the `event_id` option essentially acts like the following.

meta_content: " %sagan%: ", {id}, {id}, {id}...; meta_depth: 10;

`event_id` does this because most Windows agents (NXLog, etc) put the "event ID" at the beginning of the message.

The the data that is being processed is JSON and an "event ID" is found and properly mapped, the JSON data is used. For more information about using Sagan with JSON data, see `Sagan & JSON`.

13.17 external

`external``: {path/and/program};`

When a signature triggers with the `external` option, the `external` target is executed. The `external` program can be in any language you desire. Data is passed from Sagan via `stdin` to the `external` program. The information that is passed is the signature ID, the message (`msg`), the `classtype`, drop, `priority`, data, time, source IP, source port, destination IP, destination port, facility, syslog priority, liblognorm JSON and the syslog message.

external: /usr/local/bin/myprogram.py

13.18 syslog_facility

`syslog_facility``: {syslog facility}`

Searches only messages from a specified facility. This can be multiple facilities when separated with an '|' (or) symbol.

facility: daemon;

13.19 flexbits

`flexbits``: set, `**`{flexbit`**` name}, `**`{expire`**` time};`

Note: `flexbits` are similar to `xbits` but can deal with more complex conditions (tracking ports, reverse direction tracking, etc). However, in most cases you'll likely want to use `xbits` which are more simple and are likely to do what you need.

The `flexbits` option is used in conjunction with `unset`, `isset`, `isnotset`. This allows Sagan to "track" through multiple log events to trigger an alert. For example, lets say you want to detect when "anti-virus" has been disabled but is not related to a system reboot. Using the flexbit set you can turn on a flexbit when a system is being rebooted. Our flexbit set would look like this:

flexbits: set, windows_reboot, 30;

We are "setting" a flexbit named "windows_reboot" for 30 seconds. This means that the "windows_reboot" flexbit will "expire" in 30 seconds. The flexbit set automatically records the source and destination of the message that triggered the event. It is important to point out, the source and destination addresses are what Sagan has normalized through parse_src_ip, parse_dst_ip or liblognorm.

flexbits: {unset|isset|isnotset},{by_src|by_dst|both|reverse|username|none},{flexbit name}

This option works in conjunction with the flexbit set option. In the flexbit set example above, we are trying to detect when a system's "anti-virus" has been disabled and is not related to a system reboot. If Sagan detects a system reboot, it will set flexbit "windows_reboot". Another rule can use the presence, or lack thereof, to trigger an event. For example:

flexbits: isnotset, by_src, windows_reboot;

This means, if the "windows_reboot" flexbit is not set (ie - it did not see any systems rebooting), trigger an event. The by_src tells Sagan that the trigger (isnotset) is to be tracked by the "source" IP address. by_src, by_dst, both and none are valid options.

More examples:

flexbits: isset, both, myflexbit;

If the flexbit "myflexbit" "isset", then trigger an event/alert. Track by the source of the log message.

flexbits: isnotset, both, myflexbit;

If the flexbit "myflexbit" "isnotset", then trigger an event/alert. Track by both the source and destination of the message.

flexbits: unset, both, myflexbit;

This unset removes a flexbit from memory. In this example, unset is removing a flexbit "myflexbit" if the source and destination match (both).

Example of flexbit use can be found in the rules https://wiki.quadrantsec.com/twiki/bin/view/Main/5001880 and https://wiki.quadrantsec.com/twiki/bin/view/Main/5001881 . The first rule (5001880) "sets" a flexbit is a Microsoft Windows account is "created". The second rule (5001881) alerts an account is "enabled", but the flexbit has not (isnotset) set. In this example, it's normal for a user's account to be "created and then enabled". However, there might be an anomaly if an account goes from a "disabled" and then "enabled" state without being "created".

flexbits: {noalert|noeve}

This tells Sagan to not record certain types of data with `flexbits` when a condition is met. For example, you might not want to generate an alert when a `xbits` is `set`.

13.20 flexbits_pause

> **`flexbits_pause:`** `{seconds};`

This tells the flexbit `isset` or `isnotset` to 'wait' for a specified number of seconds before checking the flexbit state. flexbits_upause ——————

`flexbits_upause:` `{microseconds};`

This tells the flexbit `isset` or `isnotset` to 'wait' for a specified number of microseconds before checking the flexbit state.

13.21 json_content

`json_content:` `".{key}", "{search}";`

This functions similar to `content` but works on JSON key/value data. This option does _not_ depend on JSON mapping and can be used on any located key. For example:

json_content: ".sni", "www.quadrantsec.net";

Similar to `content`, the not operator (!) can also be used:

json_content:! ".sni", "www.google.com";

13.22 json_contains

`json_contains;`

Normally `json_content` will search for a literal match to a key/value pair. The `json_contains` makes the previous `json_content` do a full string search for a value. For example:

json_content: ".name", "example"; json_contains;

This will search the key "name" for the word "example". Without the `json_contains` the search is a literal match. With the `json_contains` rule option, it will search for the presences of "example" within "name". For example, with `json_contains`, this would trigger on terms like "this is an example of data" or "example test". Without the `json_contains`, it would not trigger because it would be a literal search.

13.23 json_decode_base64

... option: json_base64_decode

This decodes the base64 data from the previous `json_content`. For example:

json_content: ".payload", "BOB"; json_base64_decode;

The `json_base64_decode` keyword will base64 decoded data from the '.payload' field of the previous `json_content`. The `json_content` keyword logic will be applied to the decoded base64 data. This means that if after decoding the ".payload" key's value is the word "BOB" are found, it will constitute as a hit.

13.24 json_decode_base64_pcre

... option: json_base64_decode_pcre

This decodes the base64 data from the previous `json_pcre`. For example:

json_pcre: ".payload", "/BOB/"; json_base64_decode_pcre;

The `json_base64_decode_pcre` keyword will base64 decoded data from the '.payload' field of the previous `json_pcre`. The `json_pcre` keyword logic will be applied to the decoded base64 data. This means that if after decoding the ".payload" key's value is the word "BOB" are found, it will constitute as a hit.

13.25 json_decode_base64_meta

... option: **json_base64_decode_meta_**

This decodes the base64 data from the previous `json_meta_content`. For example:

json_meta_content: ".payload", BOB,FRANK,MARY; json_base64_decode_meta;

The `json_base64_decode_meta` keyword will base64 decoded data from the '.payload' field of the previous `json_meta_content`. The `json_meta_content` keyword logic will be applied to the decoded base64 data. This means that if after decoding the ".payload" key's value is the words "BOB", "FRANK" or "MARY" are found, it will constitute as a hit.

13.26 json_nocase

`json_nocase;`

This makes the previous `json_content` case insensitive (similar to the `nocase` option for `content`).

13.27 json_pcre

`json_pcre:` `".key"`**`, "/regularexpression/";`**

This functions similar to `pcre` but works on JSON key/value data. This option does _not_ depend on JSON mapping and can be used on any located key. For example:

json_pcre: ".sni", "/www.quadrantsec.com/i";

13.27.1 json_map

`json_map:` `"internal_value",` **`"key";`**

json_map: "src_ip", ".ClientIP";

The above example would map Sagan's internal "src_ip" to what JSON value is stored in the key "ClientIP".

This maps a JSON "key" to an internal Sagan engine value. This can be useful to map a JSON value so it can be used with other keywords like `after`, `threshold`, `bluedot`, `blacklist`, `flexbits`, `xbits`, `zeekintel`, etc. Valid "internal" Sagan values are:

`src_ip` - Maps to the internal "source" TCP/IP address. Once mapped, the data from the JSON can be used with keywords such as `after`, `threshold`, `xbits`, `flexbits`, `bluedot`, `country_code`, `blacklist`, `zeekintel`.

`dest_ip` - Maps to the internal "destination" TCP/IP address. Once mapped, the data from the JSON can be used with keywords such as `after`, `threshold`, `xbits`, `flexbits`, `bluedot`, `country_code`, `blacklist`, `zeekintel`.

`src_port` - Maps to the internal "source" TCP/IP port. Once mapped, the data from the JSON can be used with `flexbits` keyword.

`dest_port` - Maps to the internal "destination" TCP/IP port. Maps to the internal "source" TCP/IP port. Once mapped, the data from the JSON can be used with `flexbits` keyword.

`message` - Replaces existing "syslog" message with the value within the specified key. Once mapped, the JSON value can used with keywords like `parse_src_ip`, `parse_dst_ip`, `pcre`, `content`, `meta_content`, etc.

`program` - Replaces existing "program" message with the value within the specified key. Once mapped, the JSON value can be used with the rule keywords like `program` and `event_type`.

`event_type` - This is an alias for `program`.

`event_id` - Maps to the internal "event_id" value. Once mapped, the JSON value can be used with the `event_id` rule keyword.

`md5` - Maps to internal MD5 value. Once mapped, the JSON value can be used with the keywords `bluedot`, `blacklist` and `zeekintel`.

`sha1` - Maps to internal SHA1 value. Once mapped, the JSON value can be used with the keywords `bluedot`, `blacklist` and `zeekintel`.

`sha256` - Maps to internal SHA256 values. Once mapped, the JSON value can be used with the keywords `bluedot`, `blacklist` and `zeekintel`.

`filename` - Maps to internal 'filename' value. Once mapped, the JSON value can be used with the keywords `bluedot`, `blacklist` and `zeekintel`.

`hostname` - Maps to internal 'hostname' value. Once mapped, the JSON value can be used with the keywords `bluedot`, `blacklist` and `zeekintel`.

`url` - Maps to internal 'URL' value. Once mapped, the JSON value can be used with the keywords `bluedot`, `blacklist` and `zeekintel`.

`proto` - Maps to internal 'proto' value. The key is expect to be "UDP", "TCP" or "ICMP". The value is can insensitive.

13.28 json_meta_content

`json_meta_content``: ".key",` **`value1,value2,value3... ;`**

This functions similar to `meta_content` but works on a JSON key/value data. This option does _not_ depend on JSON mapping and can be used with any located key. For example:

json_meta_content: ".threat",medium,low;

This function can also be used with the not (!) operator.

json_meta_content: !".threat",informational,low;

13.29 json_meta_nocase

`json_meta_nocase;`

This makes the previous `json_meta_content` case insensitive (similar to the `nocase` option for `content`).

13.30 json_meta_contains

`json_meta_contains;`

This is similar to `json_contains` but works on the `json_meta_content` rule option.

Normally `json_meta_content` will search for a literal match to a key/value pair (strcmp). This option makes the previous `json_meta_content` do a full string search for the value (strstr).

13.31 syslog_level

`syslog_level``: {syslog level};`

Searches only messages from a specified syslog level. This can be multiple levels when separated by a '|' (or) symbol.

level: notice;

13.32 meta_content

meta_content: `"string %sagan% string",$VAR;`

This option allows you to create a content like rule option that functions with variable content. For example, let's say you want to trigger on the strings "Username: bob", "Username: frank" and "Username: mary". Without meta_content, this example would require three separate rules with content keywords. The meta_content allows you to make one rule option with multiple variables. For example:

meta_content: "Username|3a| %sagan%", $USERS;

Note: The |3a| is the hexadecimal representation of a ':' .

The %sagan% variable is populated with the values in $USERS. To populate the $USER variable, the sagan.conf would have the following variable declaration:

var USERS [bob, frank, mary]

If Sagan detects "Username: bob", "Username: frank" or "Username: mary", an event will be triggered.

Like content the ! can be applied. The ! is a "not" operator. For example:

meta_content:!"Username|3a| %sagan%", $USERS;

This will only trigger an event if the content is not "Username: bob", "Username: frank" or "Username: mary". That is, the content must not have any of the values.

The %sagan% portion of meta_content is used to specify "where" to put the $USERS defined variable. For example:

meta_content: "Username|3a| %sagan% is correct", $USERS;

Will look for "Username: bob is correct", "Username: frank is correct" and "Username: mary is correct".

13.33 meta_depth

meta_depth: `{depth value}`

Functions the same as depth for content but for meta_content. The meta_depth keyword allows the rule writer to specify how far into a log line Sagan should search for the specified patterns from a given offset.

For example, if $VAR is set to "mary, frank, bob":

meta_content: "%sagan%", $VAR; meta_depth: 10;

This would start searching at the beginning of the log line (default **meta_** offset: 0) and search only 10 bytes deep for the term "mary", "frank" or "bob".

Example with offset and depth used together:

meta_content: "bob"; meta_offset: 5; meta_depth: 10;

Sagan will start searching for the term "mary", "frank" or "bob" when it gets to 5 bytes into the log line (see meta_offset). It will only search for "mary", "frank" or "bob" after the offset for 10 bytes.

This function is identical to Snort's "depth" rule option. For more information see: http://blog.joelesler.net/2010/03/offset-depth-distance-and-within.html

13.34 meta_distance

meta_distance: {distance value}

Functions the same as distance for content but for meta_content. The meta_distance keyword allows the rule writer to specify how far into a log line Sagan should ignore before starting to search for the specified patterns relative to the end of the previous pattern match.

For example, if $VAR1 is set to "GET" and "POST" and $VAR2 is set to "download" and "upload":

meta_content:"%sagan%", $VAR1; meta_depth: 4; meta_content:"%sagan%", $VAR2; meta_distance:10;

This will cause Sagan to look for the word "GET" or "POST" within the first 4 bytes (meta_depth) of the log line. The next meta_content will start looking for the term "download" or "upload" 10 bytes away from the previous meta_depth. The above would match on the term "GET /content/downloads" but not "GET /download". The " /content/" (10 bytes) is skipped over in the distance.

This function is identical to Snort's "distance" rule option. For more information see: http://blog.joelesler.net/2010/03/offset-depth-distance-and-within.html

13.35 meta_offset

meta_offset: {offset value};

Functions the same as offset for content but for meta_content. The meta_offset keyword allows the rule writer to specify where to start searching for a pattern within a log line. This is used in conjunction with content.

For example, $VAR is set to "mary", "frank" and "bob".

meta_content: "%sagan%", $VAR; meta_offset: 5;

This informs meta_content to start searching for the term "mary", "frank" or "bob" after it is 5 bytes into the log line.

This function is identical to Snort's "offset" rule option. For more information see: http://blog.joelesler.net/2010/03/offset-depth-distance-and-within.html

13.36 meta_nocase

This makes the previous meta_content option case insensitive.

meta_content: "Username: ", $USERS; meta_nocase;

If $USERS is populated with "bob", "frank" and "mary", meta_content will ignore the case. That is, "Username: mary" and "Username: MARY" will be detected. Without the meta_nocase, meta_content is case sensitive.

13.37 meta_within

meta_within: {within value};

Functions the same as within for content but for meta_content. The within keyword is a meta_content modifier that makes sure that at most N bytes are between pattern matches using the meta_content keyword.

For example, $VAR1 is set to "GET" and "POST", while $VAR2 is set to "downloads" and "uploads";

meta_content:"%sagan", $VAR1; meta_depth:4; meta_content:"%sagan%", $VAR2; meta_distance:10; meta_within:9;

The first meta_content would only match on the world "GET" or "POST" if it is contained within the first 4 bytes of the log line. The second meta_content looks for the term "downloads" or "uploads" if it is a meta_distance of 10 bytes away from the meta_depth. From the meta_distance, only the first 9 bytes are examined for the term "downloads" or "uploads" (which is 9 bytes).

This function is identical to Snort's "within" rule option. For more information see: http://blog.joelesler.net/2010/03/offset-depth-distance-and-within.html

13.38 msg

`msg: "human readable message";`

The "human readable" message or description of the signature.

msg: "Invalid Password";

13.39 nocase

`nocase`

Used after and in conjunction with the "content" option. This forces the previous content to search for the {search} string regardless of case.

content: "sagan"; nocase;

This would search for the term "sagan" regardless of its case (ie - Sagan, SAGAN, etc).

13.40 normalize

`normalize;`

Informs Sagan to "normalize" the syslog message using the LibLogNorm library and Sagan "rulebase" data.

13.41 offset

`offset: {offset value};`

The offset keyword allows the rule writer to specify where to start searching for a pattern within a log line. This is used in conjunction with content.

For example:

content: "bob"; offset: 5;

This informs content to start searching for the term "bob" after it is 5 bytes into the log line.

This function is identical to Snort's "offset" rule option. For more information see: http://blog.joelesler.net/2010/03/offset-depth-distance-and-within.html

13.42 parse_dst_ip

`parse_dst_ip: {destination position}`

Uses Sagan's dynamic IP parsing to locate the "destination" address within a syslog message.

parse_dst_ip: 2;

The second IP address found within the syslog message will be used as the destination address. This is useful when LibLogNorm fails, is too difficult to use, or the syslog message is dynamic.

13.43 parse_port

`parse_port;`

Attempts to determine the "source port" used from the contents of a syslog message. For example, Bind/DNS messages look something like; "client 32.97.110.50#22865". The "22865" is the source port. Sagan will attempt to extract and normalize this information.

13.44 parse_proto

`parse_proto;`

Attempts to determine the protocol in the syslog message. If the syslog message contains terms in the "protocol.map" (for example, ICMP, UDP, TCP, etc), Sagan assigns the protocol to the assigned value. See fields assigned as "message" in the protocol.map.

13.45 parse_proto_program

Attempts to determine the protocol by the program generating the message. Values are assigned from the "protocol.map" (program fields). For example, if the program is "sshd" and the parse_proto_program option is used, TCP is assigned.

13.46 parse_hash

`parse_hash: {md5|sha1|sha256};`

Parses a hash out of a log message.

parse_hash: sha256;

13.47 parse_src_ip

`parse_src_ip: {source position};`

Uses Sagan's dynamic IP parsing to locate the "source" address within a syslog message.

parse_src_ip: 1;

The first IP address found within the syslog message will be used as the source address. This is useful when LibLog-Norm fails, is too difficult to use, or the syslog message is dynamic.

13.48 pcre

pcre: `"{regular expression}"`

"Perl Compatible Regular Expressions" (pcre) lets Sagan search syslog messages using "regular expressions". While regular expressions are powerful, they do require slightly more CPU to use. When possible, use the "content" option.

pcre: "/broken system|breaking system/i";

Looks for the term "broken system" or "breaking system" regardless of the strings case.

13.49 priority

priority: {priority};

Sets the probity of an alert/signature.

priority: 1;

If `priority` is set, it will override the `classtype` priority.

13.50 program

program: `{program name|another program name}`

Search only message that are from the {program}. For example:

program: sshd;

This will search the syslog message when it is from "sshd". This option can be used with multiple OR's. For example:

program: sshd|openssh;

This will search the syslog message when the program that generated it is "sshd" OR "openssh".

13.51 reference

reference: `{reference name},` **`{reference`** `url}`

Sets a reference for the signature/alert. These can be pointers to documentation that will provide more information regarding the alert.

reference: url, www.quadrantsec.com;

If the signature/alert is triggered, the reference will be "http://www.quadrantsec.com".

reference: cve,999-0531;

Will lookup CVE 999-0531 from http://cve.mitre.org/cgi-bin/cvename.cgi (from the `references.config` file).

13.52 rev

rev: {revision number};

Revision number of the rule. Increment this when a rule is changed.

rev: 5;

Revision number 5 of the rule.

13.53 sid

sid: {signature id};

"sid" is the signature ID. This has to be unique per signature.

sid: 5001021;

Sagan signatures start at 5000000. To view the "last used" signature, see https://github.com/beave/sagan-rules/blob/master/.last_used_sid

13.54 syslog_tag

syslog_tag: {syslog tag};

Informs Sagan to only search syslog messages with the specified tag. This can be multiple tags when separated with an ’|’ (or) symbol.

tag: 2d;

13.55 threshold

threshold: type {limit|suppress}, **track** {by_src|by_dst|by_username|by_string}, **count** {numbe

This allows Sagan to threshold alerts based on the volume of alerts over a specified amount of time.

threshold: type suppress, track by_src, count 5, seconds 300;

Sagan will suppress the amount of alerts via the source IP address if they exceed a count of 5 within a 300 second (5 minute) period. Every time an event happens that meets the threshold criteria, Sagan’s internal timer for this threshold will be reset. This means that the event will _not_ trigger again until the alert criteria has stopped for at least a 300 second period. If the event does stop for greater than 300 seconds, the threshold will generate 5 events and the process will start over. An example usage might be for a "brute force" attack. Lets say that the attacker is attempting 10000 passwords every second. Only the first 5 attempts would generate an alert. The threshold would apply to the remaining 9995 attempts. After the attacker tries 10000 passwords, they take a break for 20 minutes. At this point, the "suppress" threshold would time out. This means that if the attackers starts another "brute force" attack, it would trip off a maximum of 5 alerts and start thresholding again.

You can also ’track’ by multiple types. For example:

threshold: type suppress, track by_src&by_username, count 5, seconds 300;

The above would threshold by the source IP address and by the username.

threshold: type limit, track by_src, count 10, seconds 3600;

The above will threshold an alert after a count of 10 within 3600 seconds (1 hour). Unlike `suppress` the `limit` option does not reset Sagan's internal counter for this threshold. This means that 10 alerts will be generated every hour as long as the attack occurs.

13.56 within

within: `{within value};`

The within keyword is a content modifier that makes sure that at most N bytes are between pattern matches using the content keyword.

For example:

content:"GET"; depth:3; content:"downloads"; distance:10; within:9;

The first content would only match on the word "GET" if it is contained within the first 3 bytes of the log line. The second content looks for the term "downloads" if it is a distance of 10 bytes away from the depth. From the distance, only the first 9 bytes are examined for the term "downloads" (which is 9 bytes).

This function is identical to Snort's "within" rule option. For more information see: http://blog.joelesler.net/2010/03/offset-depth-distance-and-within.html

13.57 xbits

xbits:{set|unset|isset},{name},track `{ip_src|ip_dst|ip_pair} [,expire <seconds>];`

The `xbits` rule keyword allows you to track and correlate events between multiple logs. This is done by detecting an event and using the `set` for Sagan to "remember" an event. Later, if another event is detected, xbit can be tested via `isset` or `isnotset` to determine if an event happened earlier. For example, lets say you would like to detect when anti-virus is being shutdown but **not** if it is related to a system reboot or shutdown.

When Sagan detects a shutdown/reboot, Sagan can `set` an xbit. For this example, we will name the xbit being set 'system.reboot'. When Sagan sees the anti-virus being shutdown, Sagan can test to see if the xbit 'system.reboot' is set (`isset`) or is not set (`isnotset`). In our case, if the xbit named 'system.reboot' `isnotset`, we know that the anti-virus is being shutdown and is NOT related to a system reboot/shutdown.

Using `xbits` can be useful in detecting successful attacks. Another example would be the Sagan 'brute_force' xbit. Sagan monitors "brute force" attacks and `sets` an xbit associated to the source IP address (the 'brute_force' xbit). If Sagan later detects a successful login, we can test via the xbit (`isset`) to determine if the IP address has been associated with brute force attacks in the past.

Below is an example to set an xbit by the source IP address.

xbits: set,brute_force,track ip_src, expire 21600;

This will set an xbit named 'brute_force' by the source address. The xbit will expire in 21000 seconds (6 hours).

To check the xbit later, use the `isset` or `isnotset` condition. For example:

xbits: isset,brute_force,track ip_src;

If the xbit 'brute_force' was already set and is within the expire time, the `isset` will return "true" (and fire). The "track ip_src" on the `isset` or `isnotset` will compare the ip_src or the `isset` or `isnotset` rule with the `set` condition.

In certain situations, you may want to have a rule `unset` an xbit. This effectively "clears" the xbit. For example:

xbits: unset,brute_force,track ip_src;

In some situations, you might not want Sagan to record data when a `xbit` condition is met. For example, if you `set` an xbit, you might not want to generate an alert. To disable certain types of output, you can do this:

xbits: {noalert|noeve}

13.58 xbits_pause

`xbits_pause:` `{seconds};`

This tells the xbit `isset` or `isnotset` to 'wait' for a specified number of seconds before checking the xbit state.

13.59 xbits_upause

`xbits_upause:` `{microseconds};`

This tells the xbit `isset` or `isnotset` to 'wait' for a specified number of microseconds before checking the xbit state.

13.60 zeek-intel

`zeek-intel:` `{src_ipaddr},{dst_ipaddr},{both_ipaddr},{all_ipaddr},{file_hash},{url},{softwa`

Note: This option used to be known as "bro-intel"

This keyword allows Sagan to look up malicious IP addresses, file hashes, URLs, software, email, user names, and certificate hashes from Bro Intelligence feeds.

In order for the processors to be used, they must be enabled in your sagan.yaml file.

The following is a simple example within a Sagan rule:

zeek-intel: src_ipaddr;

This informs Sagan to look up the parsed source address from the Bro Intel::ADDR data. The parsed source address is extracted via liblognorm or parse_src_ip.

Multiple keywords can be used. For example:

zeek-intel: both_ipaddr, domain, url;

This instructs Sagan to look up the parsed source and destination from the Bro Intel::ADDR data. It also looks up the Intel::DOMAIN and Intel::URL. If any of the "zeek-intel" lookups return with a positive hit, the zeek-intel option is triggered. Consider the following example:

content: "thisisatest"; zeek-intel: src_ipaddr;

If a log message contains the term "thisisatest" but the parsed source IP address is not found in the Bro Intelligence feeds, the rule will not trigger. If the log message "thisisatest" is found and the src_ipaddr is found, the rule will trigger.

Sagan "zeek-intel" types:

```
src_ipaddr   Intel::ADDR              Look up the parsed source address
dst_ipaddr   Intel::ADDR              Look up the parsed destination address
all_ipaddr   Intel::ADDR              Search all IP addresses in a log message and␣
↪look them up
```

(continues on next page)

(continued from previous page)

```
both_ipaddr   Intel::ADDR            Look up the parsed source & destination address
file_hash     Intel::FILE_HASH       Search message content for malicious file hash
url           Intel::URL             Search message content for malicious URL
software      Intel::SOFTWARE        Search message content for malicious software
email         Intel::EMAIL           Search message content for malicious email
user_name     Intel::USER_NAME       Search message content for malicious user names
file_nasm     Intel::FILE_NAME       Search message content for malicious file names
cert_has      Intel::CERT_HASH       Search message content for malicious certificate␣
↪hashes
domain        Intel::DOMAIN          Search message content for malicious domain
```

CHAPTER 14

Sagan Peek

14.1 What is "saganpeek"

`saganpeek` is a utility that allows you to "peek" into Sagan memory. The utility reads the Sagan `mmap()` files. It displays the data Sagan is currently using for `after`, `threshold`, `flexbits` and `xbits`. This information can be useful in debugging Sagan or simply to view what values are currently in memory. Running `saganpeek` from the command line without any flags will show all "active" data in memory.

** Note: `saganpeek` will not display data in Redis. For example, if you are using Redis for `xbits` or `flexbits`, this data will not be displayed**

`saganpeek` –help flags:

```
--[ saganpeek help ]---------------------------------------------------------

-t, --type      threshold, after, xbit, track, all (default: all)
-h, --help      This screen.
-i, --ipc       IPC source directory. (default: /var/sagan/ipc)
```

14.2 Building "saganpeek"

After building Sagan, simply change into the `tools/` directory and run `make` and then `make install`.

CHAPTER 15

Sagan & JSON

15.1 Why JSON?

Sagan has traditionally been a syslog analysis and parsing engine. Over time, more and more platforms have been switching to JSON as an output option. Not just traditional syslog data sources but non-traditional sources like APIs and "cloud" platforms. The good side of this is the data becomes more structured and now has more context. Unfortunately, traditional Sagan rules weren't built to process this data.

The goal of Sagan is to keep the traditional syslog parsing in place and to add on JSON keyword rule options and functionality. Sagan is about processing log data, regardless of the source. This means that in many cases it is important for Sagan to properly handle JSON.

15.2 Different method of JSON input

Sagan can interpret JSON from two locations. From the named pipe (FIFO) or from a "syslog message".

The first methods is that Sagan reads incoming JSON data from a named pipe (FIFO). Traditionally, this data is in a "pipe" (|) delimited format. The "pipe" delimitation greatly limits the types of data Sagan can process. As of Sagan 2.0.0, Sagan can read JSON data via the named pipe. Most modern day syslog engines (Rsyslog, Syslog-NX, NXlog, etc) support JSON output. See sections *4.2. rsyslog - JSON mode <https://https://sagan.readthedocs.io/en/latest/configuration.html#rsyslog-json-modeg>*_ or 4.4. syslog-ng - JSON mode for more information about configuration of various log daemons.

With this in mind, this means that Sagan can collect data from non-syslog sources. For example, the IDS engine Suricata (https://suricata-ids.org) produces a lot of JSON data. Various security tool APIs like Cisco Umbrella, AWS Cloudtrail, CrowdStrike Falcon Cloud, etc. also generate a lot of JSON output. These all become possible "sources" for Sagan data processing.

The second method of JSON data collection is via the syslog "message" field. Some syslog "forwarders" use this method to send SIEMs data. The idea is that the data is transferred via the traditional syslog transport but the message contains the JSON data. Sagan can interpret that data for alerting purposes.

15.3 JSON "mapping"

Either method you decide to receive the JSON data in, it is likely you will want to "map" the data so that Sagan can properly process it. You can think of mapping this way; When Sagan receives JSON data, it doesn't know what is "important" and what isn't. "Mapping" allows you to assign values to the data so the engine can process it and signatures can be used. It is also important to understand that different platforms label key/value pairs differently. For example, a source IP address on one platform might be "src_ip", while on another platform it might be "source_ip". Mapping allows you to assign the "source" IP value from the JSON.

"Mapping" allows you to use signature keys words like `content`, `pcre`, `meta_content`, etc. and features like `threshold`, `after`, `xbits`, etc.

Simply put, "Mapping" allows you to assign JSON "key" data to specific internal Sagan values.

Within the Sagan rules are two files. One is `json-input.map` and the other is `json-message.map`. These are the mapping files that are used depending on your method of input. These files can be altered to support the JSON mapping you might need and come with some example mapping.

In some cases, "mapping" might be over kill and can be skipped. See `When mapping is not needed`.

15.4 How JSON nest are processed

Sagan will automatically "flatten" nests. For example, let say you want to process the

following JSON format.

{"timestamp":"2019-11-19T20:50:02.856040+0000","flow_id":1221352694083219,"in_iface":"eth0","event_type":"alert","src_i
Ping Packet [ICMP]","category":"Not Suspicious Traffic","severity":3},"flow":{"pkts_toserver":2,"pkts_toclient":0,"bytes_toserv
11-19T20:50:01.847507+0000"},"payload":"elXUXQAAAACtDw0AAAAAAE9GVFdJTkstUElOR9raU09GVFdJTkstUElOR9

All nest, including the top nest, start with a `.`. For example, the JSON key "timestamp" will become `.timestamp`

internally to Sagan. The "event_type" and "src_ip" would become `.event_type` and `.src_ip`. For nested objects like "alert", you would access the "signature_id" as `.alert.signature_id`. This structure is similar to JSON processing commands like `jq`.

There is no limitations on nest depths. This logic applies for JSON "mapping" and Sagan signature keywords like `json_content`,

`json_pcre` and `json_meta_content`.

15.5 When mapping is not needed

In most cases, you'll likely want to performing mapping for your JSON data. However, there are some instances where mapping might not be required. Keep in mind, without mapping things like `threshold`, `after`, `xbits` might not perform properly.

Regardless of whether Sagan properly maps the JSON, it will internally still split the key/value pairs in real time. While you won't be able to use the standard Sagan rule operators (ie - `content`, `pcre`, etc) you will be able use some JSON specific operators.

These are `json_content`, `json_pcre` and `json_meta_content`. With these, you can specify the key you want to process and then what you are searching for.

This can be useful when used in conjunction with mapping. This way you can use traditional Sagan keywords (`threshold`, `after`, `content`, etc) along with JSON specific (`json_content`, `json_pcre`, etc) rule options.

15.6 Mappable JSON Fields

While not all JSON field can be internally mapped, these are the Sagan internal fields that should be consider. Each field has different functionality internally to Sagan. For example, if you want to apply rule operators like `threshold` or `after` in a signature, you'll likely want to map `src_ip` and/or `dst_ip`. The following are internal Sagan variables/mappings to consider for mapping.

Fields to consider for internal JSON mappings are as follows.

`src_ip`

This value will become source IP address of the event. This will apply to rule options like `threshold`, `after`, `xbits`, `flexbits`, etc.

`dst_ip`

This value will become the destination IP address of the event. This can also be represented as `dest_ip`. This will apply to rule options like `threshold`, `after`, `xbits`, `flexbits`, etc.

`src_port`

JSON data for this will become the source port of the event. This will apply to rule options like `flexbits`.

`dst_port`

JSON data for this will become the destination port for the event. This will apply to rule options like `flexbits`. This can also be represented as `dest_port`.

`message`

The JSON for this value will becoming the syslog message. This will apply to rule options like `content`, `pcre`, `meta_content`, `parse_src_ip`, `parse_dst_ip`, `parse_hash`, etc.

`event_id`

The JSON data will be applied to the `event_id` rule option.

`proto`

This will represent the protocol. Valid options are TCP, UDP and ICMP (case insensitive).

`facility`

The JSON data will be mapped to the syslog facility. This will apply to the rule option `facility`.

`level`

The JSON data will be mapped to the internal Sagan variable level. This will apply to the rule option `level`.

`tag.`

The JSON data will be mapped to the internal Sagan variable of tag. This will apply to the rule option `tag`.

`syslog-source-ip`

The JSON data will be mapped to the internally to Sagan's syslog source. This should not be confused with `src_ip`. If `src_ip` is not present, the `syslog-source-ip` become the `src-ip`. This might apply to `threshold` and `after` is `src_ip` is not populated.

`event_type`

The JSON data extracted will be applied internally to the Sagan variable of "program". `event_type` is simply an alias for `program` and both can be interchanged. This applies to rule options like `program` and `event_type`.

`program`

The JSON data extracted will be applied internally to the Sagan variable of "program". `program` is simply an alias for `event_type` and both can be interchanged. This applies to rule options like `program` and `event_type`.

time

The JSON data extracted will be applied internally to the syslog "time" stamp. This option is recorded but is not used in any rule options.

date

The JSON data extracted will be applied internally to the syslog "date" stamp. This option is recorded but is not used in any rule options.

15.7 JSON via named pipe (FIFO)

Mapping for JSON data coming in via the named pipe (FIFO) is configured in the `sagan-core` section under `input-type`. Two types are available, `json` and `pipe`. If `pipe` is used, the sections below (`json-map` & `json-software`) are ignored.

```
# Controls how data is read from the FIFO. The "pipe" setting is the traditional
# way Sagan reads in events and is default. "json" is more flexible and
# will become the default in the future. If "pipe" is set, "json-map"
# and "json-software" have no function.::

input-type: json                       # pipe or json
json-map: "$RULE_PATH/json-input.map"  # mapping file if input-type: json
json-software: syslog-ng               # by "software" type.
```

The `json-map` function informs the Sagan engine where to locate the mapping file. This is a file that is shipped with the Sagan rule set and already has some mappings within it. The next option is the `json-software` type. The `json-input.map` typically contains more than one mapping type. The `json-software` tells Sagan which mapping to use from that file. A typically mapping for Syslog-NG looks like this:

```
{"software":"syslog-ng","syslog-source-ip":".SOURCEIP","facility":".FACILITY","level":
↪".PRIORITY","priority":".PRIORITY","time":".DATE","date":".DATE","program":".PROGRAM
↪","message":".MESSAGE"}
```

These are key/value pairs. The first option (ie - `message`, `program`, etc) is the internal Sagan engine value. The value to the key is what Syslog-NG names the key.

When Sagan starts up, it will parse the `json-input.map` for the software type of "syslog-ng". If the `software` of "syslog-ng" is not found, Sagan will abort.

When located, Sagan will expect data via the named pipe to be in the mapped JSON format. Data that is not in this format will be dropped. To understand mapping better, below is an example of JSON via the named pipe that Sagan might receive:

```
{"TAGS":".source.s_src","SOURCEIP":"127.0.0.1","SEQNUM":"437","PROGRAM":"sshd",
↪"PRIORITY":"notice","Authentication failures; logname= uid=0 euid=0 tty=ssh ruser=␣
↪rhost=49.88.112.77  user=root","LEGACY_M"dev-2","HOST":"dev-2","FACILITY":"authpriv
↪","DATE":"Jan  2 20:12:36"}
```

As we can see, Syslog-NG maps the syslog "message" field as ".MESSAGE". The Sagan engine takes that data and internally maps it to the "message" value. It repeats this through the rest of the mapping.

Mapping this way becomes a more convient and flexible method of getting data into Sagan than the old "pipe delimited" format.

Note: When processing JSON via the named pipe, only one mapping can be used at a time.

15.8 JSON via syslog message field

The mapping concept for Sagan when receiving JSON data via the syslog "message" is similar to JSON data via the named pipe.

Unlike JSON data via the named pipe, when receiving data via a syslog "message" multiple maps can be applied. The idea is that your Sagan system might be receiving different types of JSON data from different systems.

To determine which "map" works best, the Sagan engine does an internal "scoring" of each map. Sagan will then apply the best map that matches the most fields. This means that you might want to "map" fields event if you don't plan on using them. This ensures that the proper "map" will "win" (score the highest).

To enabled JSON syslog message processing, you will need to enable the following fields within the `sagan-core` part of the sagan.yaml.

```
# "parse-json-message" allows Sagan to detect and decode JSON within a
# syslog "message" field.  If a decoder/mapping is found,  then Sagan will
# extract the JSON values within the messages.  The "parse-json-program"
# tells Sagan to start looking for JSON within the "program" field.  Some
# systems (i.e. - Splunk) start JSON within the "program" field and
# into the "message" field.  This option tells Sagan to "append" the
# strings together (program+message) and then decode.  The "json-message-map"
# tells Sagan how to decode JSON values when they are encountered.

parse-json-message: enabled
parse-json-program: enabled
json-message-map: "$RULE_PATH/json-message.map"
```

The `parse-json-message` configures Sagan to automatically detect JSON within the syslog "message" field. The `parse-json-program` configures Sagan to automatically detect JSON within the syslog "program" field.

Some applications will send the start of the JSON within the "program" field and it will overflow into the "message" field. The `parse-json-program` option configures Sagan to look for JSON within the "program" field and append the "program" and "message" field if JSON detected.

The `json-message-map` contains the mappings for systems that might be sending you JSON. As with the `json-input.map`, the Sagan rule sets come with a `json-message.map`.

An example mapping:

```
{ "software":"suricata", "syslog-source-ip":".src_ip","src_ip":".src_ip","dest_ip":".
↪dest_ip","src_port":".src_port","dest_port":".dest_port","message":".alert.
↪signature,.alert_category,.alert.severity","event_type":".hash","time":".timestamp",
↪"date":".timestamp", "proto":".proto" }
```

Unlike named pipe JSON mapping, the "software" name is not used other than for debugging. When Sagan receives JSON data, it will apply all mapping to found in the `json-message.map` file.

Note of the “message” field. This shows the "message" being assigned multiple key values. In this case the key “.alert.signature”,”.alert.category” and “.alert.severity” will be become the “message”. Internally to Sagan, the “message” will become “key:value,key:value,key:value”. For example, let say the JSON Sagan is processing is the follow Suricata JSON line:

{"timestamp":"2020-01-03T18:20:05.716295+0000","flow_id":812614352473482,"in_iface":"eth0","event_type":"alert","src_ip'
Ping Packet [ICMP]","category":"Not Suspicious Traffic","severity":3},"flow":{"pkts_toserver":5,"pkts_toclient":0,"bytes_toserve
01-03T18:20:01.691594+0000"},"payload":"1YUPXgAAAADM7QoAAAAAAE9GVFdJTkstUElOR9raU09GVFdJTkstUElOR

Internally to Sagan the "message" will become:

```
.alerts.ignature:QUADRANT Ping Packet [ICMP],.alert.category:Not Suspicious Traffic,
↪alert.severity:3
```

This means any signatures you are going to create will need to take this format into account. In cases where you would like the entire JSON string to become the message, simply make the "message" mapping `%JSON%`. This tells Sagan that the entire JSON string should be considered the "message".

CHAPTER 16

Journald

16.1 What is "journald"?

Journald is a system for collecting logs and data from devices running "systemd". Many distributions have moved away standard syslog services in favor of "journald". The concept is to replace standard "text" base logging for a more "database" binary logging approach.

While this method has advantages, there are several limitations. Software like "Sagan" doesn't natively read "journald" files. Journald also lacks the ability to send logs to a remote host. Journald relies on services like `syslog-ng` and `rsyslog` to send logs to a remote host. While there are some methods to send logs to a remote host via Journald, most are not mature and more of a "proof of concept" than a solution. This makes using a service like `syslog-ng` or `rsyslog` the best method to send logs generated by Journald.

16.2 Analyzing journald logs locally

Using the "Journald" command `journalctl`, it is possible to create a JSON stream representing Journald data. Using Sagan built in JSON processing, it is possible to analyze this data. As Journald writes log data, the `journalctl` converts it to JSON and sends it to `stdout`. This can be redirected to a named pipe (FIFO). For example, `journalctl -f -o json > /var/sagan/fifo/journald.fifo` will direct log data to a named pipe which Sagan can read. Within the Sagan configuration file, you would want to set the following options:

```
input-type: json                        # pipe or json
json-map: "$RULE_PATH/json-input.map"   # mapping file if input-type: json
json-software: journald                 # by "software" type.
```

16.3 Analyzing journald logs remotely

In situations where `syslog-ng` or `rsyslog` is not an option, you can using `journalctl` to send logs to a remote host in raw JSON. For example, `journalctl -f -o json | nc 192.168.1.1 1514`. This would using `netcat` to send logs to 192.168.1.1 on port 1514. Your receiver would need to be configuration to accepts incoming connection and date in a __raw__ format (non-syslog). Sagan could then be used on the receiving side to analyze data from various devices. You would likely want to wrap the "journalctl" in a script and infinite loop so `journalctl` will automatically restart if the TCP log connection is broken.

CHAPTER 17

High Performance Considerations

Depending on your hardware, Sagan can operate comfortably up to about 5k "events per/second" (EPS) using default configurations. When you hit this level and higher, there are a few configuration options to take into consideration.

17.1 batch-size

The most important thing is the `batch-size` sagan.yaml configuration option. By default, when Sagan receives a log line, the data is sent to any available thread. Due to memory protections (pthread mutex lock/unlock), this isn't efficient. The system starts to spend more time protecting the memory location of the single line of log data than processing the log line.

The `batch-size` allows Sagan to send more data to worker threads and use less "locks". For example, with a `batch-size` of 10, Sagan can send 10 times more data with only one "lock" being applied. At even higher rates, you may want to consider setting the `batch-size` to 100.

The default batch sizes are 1 to 100. On very high performance systems (100k+ EPS or more), you may want to consider rebuilding to handleeven larger batches. To do this, you would edit the *sagan-defs.h* and change the following.

```
#define MAX_SYSLOG_BATCH        100
```

To

```
#define MAX_SYSLOG_BATCH        1000
```

Then rebuild Sagan and set your `batch-size` to 1000. While you will save CPU, Sagan will use more memory. If you sent the *MAX_SYSLOG_BATCH* to 1000 and only set the `batch-size` to 100, Sagan will still allocate memory for 1000 log lines. In fact, it will do the per-thread! Think of it this way:

:: (MAX_SYSLOG_BATCH * 10240 bytes) * Threads = Total memory usage.

The default allocation per log line is 10240 bytes.

17.2 Rule sets

At high rates, consideration should be given to the rules that you are loading. Unneeded and unused rules waste CPU.

If you are writing rules, make sure you use simple rule keywords first (`content`, `meta_content`, `program`, etc) before moving to more complex rule options like `pcre`. The more simple rule keywords can be used to "short circuit" a rule before it has to do more complex operations.

Software like `Snort` attempts to arrange the rule set in memory to be more efficient. For example, when `Snort` detects multiple `content` modifiers, it shifts the shortest lenght `content` to the front (first searched). Regardless of the `content` rule keywords placement within a rule.

Because logs are inherently different than packets, `Sagan` does not do this! If you have multiple `content` keywords, `Sagan` will use them in the order they are placed in the rule. You will want to use the least matched keywords as the first `content`. For example:

::

```
# This will use more CPU because "login" is common.

content: "login"; content: "mary";

# This will use less CPU because "mary" is likely less common.

content: "mary"; content: "login";
```

The same login applied to `pcre` and `meta_content`.

17.3 Rule order of execution

Sagan attempts to use the least CPU intensive rule options first. This means that if a `Sagan` rule has multiple `content` keywords and multiple `pcre` keywords, the `content` rule keywords are processed first. If the `content` keywords do not match, then there is no need to process the `pcre` keywords. The order of execution within a rule is as follows:

The `program` field is the very first thing to be evaluated.

The `content` is the next option Sagan takes into consideration.

The `meta_content` is next.

Finally the `pcre` option, which is consided the heaviest, is the last.

CHAPTER 18

Contributing & Coding Style

18.1 How to contribute to Sagan

18.1.1 Rules & Signatures

Sagan signatures are the life-blood of Sagan! It is probably one of the most valuable ways that you can contribute to Sagan. If you understand the basics of how Suricata IDS or *Snort <https://snort.org>*_ signatures function, then you already know how to construct Sagan rules. If you want to add to a rule set or create an entirely new rule set, this is a huge way to contribute!

18.1.2 Code

Are you a C programmer and want to add some functionality to Sagan? That's great! You might want to share your idea with the Sagan coding team. This way, if it is not an idea that will fit with Sagan or it is a duplicated effort, you'll know before you dive in. The best way to contact the Sagan team is via the Sagan mailing list (https://groups.google.com/forum/#!forum/sagan-users).

Also, check the `Coding guidelines and style` section of this page.

18.1.3 Documentation

Code is great but it is almost worthless without proper documentation. Do you see something in our documentation that is incorrect? Perhaps something that could be better written or explained? Feel free to contribute!

The Sagan documentation is part of the Sagan source tree. We use the Python Sphinx system and "readthedocs.org" for publication.

** MORE ABOUT HOW TO CONTIBUTE DOCS HERE! **

18.1.4 Blogs & articles

Tell us, and better yet, the world, how you are using Sagan. We are always interested to see who and how our software is being used. In return, we will link to your articles from within our Sagan ReadTheDocs.org documentation page! This help spread the word about Sagan and we truly appreciate it!

18.2 Coding guidelines and style

18.2.1 Coding style

Sagan development is primarily done in C. We use the `gnu` "artistic style". If you are not familiar with the `gnu` artistic style, that is okay. We use tools like `astyle` to keep code consistent. Using tools like `astyle` allows you to write code in the style you are most comfortable with and then convert it before committing. In fact, it is pretty rare that the main contributors manually follow these guidelines!

To install `astyle`, as root:

```
apt-get install astyle
```

Before committing your code, simply run the following command within your source tree:

```
astyle --style=gnu --suffix=none --verbose *.c *.h
```

18.2.2 Coding Guidelines

While everyone has their own set styles and methods of coding, there are a few things that we prefer to see in the Sagan code. The biggest thing is consistency. Not only by the coding "style" (see `Coding Style`) but also logical formatting.

Consistency with "if" statement is required. For example:

```
/* Incorrect */

if (0 == variable )
   {
   ...
   }
```

Will be rejected. The proper coding format with Sagan would be:

```
/* Correct */

if ( variable == 0 )
   {
   ...
   }
```

When using boolean operators, be sure and use the `stdbool.h` `true` and `false`. For example:

```
/* Correct */

if ( variable == true )
   {
```

(continues on next page)

(continued from previous page)

```
   ...
   }

/* Incorrect */

if ( variable == 1 )
   {
   ...
   }
```

Your code should contain comments that are clear. Proper comment syntax is desired as well. For example:

```
// Example incorrect comment

if ( x == y )          /* Incorrect comment */
   {
   ...
   }

/* Example correct comment */

if ( x == y )          // This is acceptable
   {
   ...
   }
```

The `{` and `}` are converted in the GNU "artistic style". Even if you do not prefer this formatting, programs like `astyle` can correct them before you commit. For example:

```
/* Incorrect */

if ( x == y ) {
   ...
   }

/* Correct */

if ( x == y )
   {
   ...
   }

/* Incorrect */

if ( x == y )
     b = a;

/* Correct */

if ( x == y )
   {
     b = a;
   }
```

These are a few simple rules to consider before contributing code. In many cases `astyle` will address them for you.

CHAPTER 19

Sagan Blogs

19.1 Dynamic Rules with Sagan.

Posted by Champ Clark on November 14, 2016

One of the biggest problems faced with log monitoring is ensuring that the proper rules are loaded. Just like with packet based IDS systems, during the installation and setup process, you typically enable the rules that you think are relevant to your environment. The problem is, environments change over time and we might neglect to go back and determine if the original rules we enabled are still relevant. The idea behind "dynamic rules" is to detect changes in the logging infrastructure and make adjustments by "dynamically" loading rules and letting you and your staff know.

It is pretty common for networks to change over time. For example, let's say that during deployment of Sagan in your network it was analyzing Linux, Windows, and Palo Alto firewall logs. Two years later, your organization decides to replace its Palo Altos with Cisco ASA firewalls; have you made the appropriate changes to your monitoring infrastructure to take into account the Palo Alto to Cisco ASA Switch? It's an easy thing to forget and miss.

The idea is to have Sagan "see" the changes and "dynamically" load the rules and alert you to the fact.

To detect the change, we have created a "dynamic.rules" rule set that utilizes the power of the Sagan rule structure. The idea is that we can create rules that will "detect" when Sagan "sees" new logs entering the system. The "dynamic.rules" watches for characteristics of various log types and when they are detected, responds by loading the rules and alerting your staff.

One thing we don't want to do is take away CPU cycles from normal analysis to detect "new" logs. Think of it this way, the more "signatures" you feed Sagan, or any IDS system, the more CPU it takes to process data through them. Increasing your total signature size increases your load.

We have gotten around the CPU load problem by creating a "sample" rate. We don't necessarily want to examine every log received to determine if it's "new" to the system or not. With a "sample" rate, we tell Sagan to only examine every X log for "new" content. This is done by utilizing the "dynamic_load" processor with the "dynamic.rules". The "processor" line looks like this in your "sagan.conf":

processor dynamic_load: sample_rate=100 type=dynamic_load*****

The sample_rate is set to 100. This means that every 100th log line received, Sagan will examine it for "new" characteristics. If the log line is determined to be "new" to the system via the "dynamic.rules", dynamic_load (via the "type=") tells Sagan to load the associated rule set. Possible options for "types" are dynamic_load, which logs and writes a unified2 record and loads the associated rule set. The log_only type tells Sagan to simply write out to the sagan.log file that it has detected a new log type. The alert tells Sagan to create a single unified2 record (an alert) that it has detected a new log type.

The use of the sample_rate greatly reduces the CPU load and allows for the amount of fine-tuning that you feel comfortable with. A sample_rate of 100 means you'll use 1/100 CPU time for new log detection. You could increase the sample_rate but then it might take longer to detect "new" logs entering the system. Alternatively, you could decrease the sample_rate, which will detect new logs entering Sagan faster, but use more CPU.

For the time being and for the purposes of our testing, a default of 100 seems to be a good starting place.

Now that we've determined the amount of data we want to process for "new" logs, let's look into an example of "dynamic.rules":

alert syslog $EXTERNAL_NET any -> $HOME_NET any (msg: "[DYNAMIC] Cisco ASA logs detected via program."; program: %ASA*|%FWSM*; dynamic_load: $RULE_PATH/cisco-pixasa.rules; classtype: dynamic-rules; reference: url,wiki.quadrantsec.com/bin/view/Main/5002967; sid:5002967; rev:2;)

Note the new dynamic_load rule option. This tells Sagan that this is a "dynamic" rule that should follow the configurations set by the "dynamic_load" processor. It also informs Sagan "what" to load when a "new" log type is detected. Note that you can use sagan.conf configuration variables within the rule (i.e. - $RULE_PATH).

The rest of the rule works like a normal Sagan rule. In this simple example, we know that Cisco ASA's typically uses the "program" of %ASA-{number-code}. If Sagan sees a log line with a program of %ASA-* and Sagan has not previously loaded the "cisco-pixasa.rules", it will automatically load them and trigger a log/unified2 alert.

One interesting result we've seen in testing is using "dynamic.rules" to tell the user what rules to load! For example, we could start Sagan without any normal non-dynamic rules enabled. That is, the only rules enabled would be "dynamic.rules". Sagan could then inform the user what rules it would load. With that data, the user could manually load those and other associated rules (geoip rules, malware rules, etc).

Detection of changes to infrastructure is very important. Using "dynamic.rules" allows you to detect those changes quickly and automatically adjust.

19.2 What the Sagan Log Analysis Engine Is... and What It Is Not.

Posted by Champ Clark on August 22, 2016 Article by Champ Clark III.

With so many log analysis tools out there, we sometimes see strange comparisons between the Sagan log analysis engine and unrelated tools. For example, people often ask how Sagan compares to Splunk. In our opinion, these are two different tools with two different jobs.

For one, the Sagan log analysis engine is a tool that was programmed completely with a focus on security. Splunk and similar tools, on the other hand, are analysis and log archival search utilities with security focused functionality added on later. We aren't suggesting that this is a bad thing, and it doesn't mean that Splunk and similar tools are "bad." But, as security tools they are attempting to accomplish different goals.

If anything, Sagan is more similar to tools like OSSEC, rather than Splunk.

What we are doing with Sagan is trying to detect the successful "bang" (attack) when it occurs. Robert Nunley turned me on to this military terminology some time ago and I think it applies to information security very well.

"To think about an attack on a timeline, bang is in the middle. Bang is the act. Bang is the IED explosion, the sniper taking a shot, or the beginning of an ambush." (From the book "Left of Bang"; https://www.amazon.com/Left-Bang-Marine-Combat-Program/dp/1936891301)

"Left of bang" is before the attack has occurred. The "bang" is the time of the attack and where Sagan does its best detection. Retro or non-real time detection of an attack is at the "right of bang," where most log analysis tools operate today.

> At Quadrant, we are working with the "bang" and at the "right of bang." Using technologies that operate at both time points allows our SOC to detect threats better.

Operating on the "left of bang" is more difficult to accomplish. We are proactively working to improve this within our BlueDot threat intelligence (part of Sagan), and this is also where projects like Quadrant's new "APT Deflector" (patent pending) come into play.

The idea behind Sagan is for it to treat logs similarly to how Snort (IDS) treats packets, in rapid, real- time analysis and correlation. Let's examine these two statements.

Snort (IDS) and "Full Packet Capture" (FPC) have two different functions. If I need to search for something in my FPC archive, I can. I put IDS in front so that it might detect "bad things" happening before I have to go into my FPC archive.

Sagan and log archival have two different functions. If I need to search for something in my log archive, I can. I put Sagan in front so that it might detect "bad things" happening before I have to go into my log archive.

Sagan is the IDS for logs, FPC is the "log archive."

In some cases, Sagan is able to tell you enough about an attack, so that you might not need to dig any further. In other cases it does not. Instead, you use the Sagan data to point you in the right direction to use with other tools.

As a "technology + people" company, this is exactly how we use Sagan at Quadrant Information Security in our SOC. When IDS detects a "bad thing" our SOC handlers might utilize "Bro" (https://www.bro.org/) or FPC to get a clearer picture of what is going on. When Sagan detects a "bad thing" happening, our SOC handlers use raw log searches to paint a better picture about what is going on. We then relay that clear picture back to our clients.

Sagan is also intended to be the "glue" between security devices. I just recently had a friendly argument with the author of Snort, Martin Roesch, about something he said in his RSA keynote speech "Advanced Strategies for Defending against a New Breed of Attacks" (The full video is at: https://www.youtube.com/watch?v=O_mmGUu_6gM . At 9:00 minutes you get to the points I'm referring too).

It seemed to me while watching his video that he was suggesting that we come up with a means that would allow different security devices to communicate with each other. It sounded a lot like "Security Device Event Exchange" (SDEE) (https://en.wikipedia.org/wiki/Security_Device_Event_Exchange) to me. He also stated that he believed this couldn't be accomplished at the log level.

My counter-argument is that vendors are never going to "work together", sing "kumbaya" and start using a standard, unified format. It's been tried, multiple times, and each time it has failed. What are the odds that Cisco C-level executives are going to want to see data interaction and exchange with say, Juniper gear? Or Fortinet?

Speaking to his second point, in an ideal world, your Linux servers would be able to share "security" related information with your Microsoft servers. For example, let's say that an attacker is attempting to 'brute force' your Linux server's SSH service. Let's also say that the brute force was unsuccessful. One hour later, a valid successful login via Microsoft RDP is detected from the same IP address. This might be something you want to investigate.

This is exactly what Sagan does, at the log level. While the Linux and Windows servers won't "share" information, since they both send data back to Sagan, Sagan becomes the intermediary for the data. Another example might be your IDS detecting an SQL injection attack, but your "Web Application Firewall" (or mod_security) blocks the attack. We might want this data, but not escalate it to a phone call at 3:00 a.m. We can now also "track" the attacker across our network.

The idea is to do this in real-time. Not retro-actively hours or days later.

We do this in Sagan with what is known as "flowbits". Robert Nunley from Quadrant wrote an excellent post some time ago about flowbits (https://quadrantsec.com/about/blog/sagan_flowbit/). The next thing that's usually said is, "ah, but now I have to figure out how to write rules with flowbits." Actually, we've already written many rules with flowbits of common scenarios, just like the examples above, and we are constantly improving our rule set. However, you also have the power to write your own rules.

The idea behind the Sagan log analysis engine is to be a real-time "IDS" for your logs. It is the "glue" between your devices.

There is no single tool that is a silver bullet and anyone claiming that there is, is lying.

19.3 Sagan 1.0.0 log analysis engine released!

Posted by Champ Clark on October 23, 2015

In June 2010, we completed initial work on Sagan 0.0.1 which was a very basic outline of what we thought a real-time log analysis engine should be. Historically, people treated logs as an archive of only the past activities, and in 2010, many solutions for "log analysis" were based on command line tools and concepts like grep. This approach is fine and certainly useful, but why was real-time log analysis not really a "thing?" We never suggested getting rid of historical log search functionality, but the lack of "real time" detection was troubling; we expect some security software, like Intrusion Detections Systems (IDS) to be "real time," so why was log analysis not treated the same way? After all, if someone told you that their solution to packet inspection was to "look at all the packets via a 'grep' every Friday," you would laugh at them. We decided to tackle this problem because of our own selfish needs.

When we started developing Sagan, we naturally focused on our own needs at Quadrant Information Security. Since we are an MSSP (Managed Security Service Provider), we needed to be able to monitor security appliances and software similarly to how we monitored our "Snort" instances. Back in 2010, pre-Cisco/Sourcefire buy-out, not all companies were interested in Snort. They "trusted" more "mainstream" products from companies like Cisco, Sonicwall, Fortinet, etc. As much as we argued that Snort was a better IDS/IPS solution, many potential customers simply were not interested; "we're a Cisco shop, that's the way it is," we heard this a lot.

Initial development began so that we, as an MSSP, could say "yes, we can monitor that." At the time, that was our primary need, which meant that Sagan had to be 100% real time. It would not be reasonable for our analysts to have to "grep" logs daily in order to search for possible malicious activity. Software should be able to provide this data and do it better. To be real-time in environments with mass amounts of log data, Sagan needed to be multi-CPU aware and memory-efficient. Therefore, we designed Sagan in C using threading (pthreads). If your analysis platform has multiple CPUs and/or cores, Sagan would need to "spread" the log analysis load across them. Since our analysts already understood packet analysis via Snort rules, it made sense to have Sagan use a similar syntax, which also meant that Snort rule management software like "pulledpork" would inherently work with Sagan.

Since we were already traveling down the "very much like" Snort path in terms of design, we decided that we might as well adopt the Snort "unified2" output format, which means that Sagan can store its data in the same place that Snort does. This also meant that we can correlate log events to our packet events, and that we are out-of-the-box compatible with Snorby, BASE, Squil, etc.

Overall, those were the basic milestones we wanted to get to. As time went on, Sagan required more complexity that was not foreseen at the time of its inception (i.e., flowbits). In August of 2015, after 5 years of development, we put Sagan into a "code freeze" which means that rather than trying to add complex new features to Sagan, we focus on stability. And although Sagan has always been pretty stable, we started testing across a lot of platforms that varied in log data flow, rules enabled, and environmental complexity. In August of 2015 released "RC1" (Release candidate #1) to the public to help us test Sagan. We made it up to "RC5", and today, October 23rd, 2015, we're proud to call this Sagan 1.0.0.

Today, Sagan is used around the world by medical companies, hospitals, banks, credit unions, financial institutions, petroleum companies, law firms, supermarket chains, telecommunications companies, accounting firms, manufacturers, hosting providers, insurance companies, colleges, universities and various law enforcement agencies. It is even used by other network and computer security companies, and these are just the organizations that we know use Sagan!

We are very proud of how far Sagan has come since its inception. Sagan is a complex piece of software that required the input and help from many people. I like to highlight that fact since Sagan would not be where it is today had it not been for all of these people willing to spend time deep in the Sagan code, and developing rules. If you have a moment, please check out the contributors via the "sagan –credits" flag or https://github.com/beave/sagan/blob/master/src/sagan-credits.c

Now that 1.0.0 is behind us, we look forward to adding some new "killer" functionality. It is going to be a really fun ride. Check out the open source version of Sagan at http://sagan.io

19.4 Sagan output to other SIEMs

Posted by Champ Clark on November 06, 2014

Sagan is a very powerful engine at detecting threats contained in log data. When Sagan detects something that it believes you should know about, it can "output" these alerts in several formats. The most popular and useful of these output formats is "Unified2". Unified2 is typically used by Snort, Suricata and Sagan to record details about an event/alerts. It records not only the payload, or in Sagan's case, the offending log message but other details as well. The source, destination IP address, source and destination ports and much more.

What makes this output format so powerful is that it gives Sagan the ability to put event and alert data in the same location as other utilities like Snort and Suricata. This means you can view "threats" from "one pane of glass" (one console). So instead of having IDS/IPS threats in one console and Sagan log analysis data in another, it all gets stored in a unified location. With that said, there are power instances you might want to correlate more than just "threat" data. For example, you might was to send this data to a centralized log server. If you are sending your Snort/Suricata data to a centralized log server, then it likely makes sense you would like to do the same with Sagan data.

This give you the ability to not only look at the threat data from Snort, Suricata and Sagan, but other data "surrounding" the event.

To do this, we use Sagan's "syslog" output format. This lets Sagan send events and alerts to the systems "syslog" facility. These can then be forwarded to our centralized log server and/or SIEM. As we've stated in pervious blog posts, we try to maintain some compatibilty with Snort in some respects. This allows Quadrant Information Security to work on creating the best log analysis engine without having to worry about things like rule management, rule formats, etc.

With this in mind, it should come as no suprise that Sagan's "syslog" output format works very similar to Snort's "syslog" output format. In your sagan.conf file, you would add the following:

output syslog: LOG_AUTH LOG_ALERT LOG_PID

These are also the default settings for Sagan. The output format in the configuration file is like this:

output syslog: (facility) (priority) (syslog options)

(Supported facilities: LOG_AUTH, LOG_AUTHPRIV, LOG_CRON, LOG_DAEMON, LOG_FTP, LOG_INSTALL, LOG_KERN, LOG_LPR, LOG_MAIL, LOG_NETINFO, LOG_RAS, LOG_REMOTEAUTH, LOG_NEWS, LOG_SYSLOG, LOG_USER, LOG_UUCP, LOG_LOCAL0, LOG_LOCAL1, LOG_LOCAL2, LOG_LOCAL3, LOG_LOCAL4, LOG_LOCAL5, LOG_LOCAL6, LOG_LOCAL7)

(Supported priorities: LOG_EMERG, LOG_ALERT, LOG_CRIT, LOG_ERR, LOG_WARNING, LOG_NOTICE, LOG_INFO, LOG_DEBUG)

(Supported options: LOG_CONS, LOG_NDELAY, LOG_PERROR, LOG_PID, LOG_NOWAIT)

With the syslog output configured, Sagan can now generate messages to your local syslog daemon that look like this:

sagan[8517]: [1:5002178:2] [OPENSSH] SSH login success after brute force attack! [Classification: Correlated Attack] [Priority: 1] {TCP} 10.10.10.10:42131 -> 10.10.10.11:22

You might be thinking to yourself how similar the Sagan syslog message looks to a Snort or Suricata syslog message. You would be correct! Sagan does this so that you might take advantage of Snort syslog parsers within your SIEM! For example, lets say you use Splunk to collect logs from your Snort IDS/IPS systems. In Splunk, you might have built a log parser to extract important data from Snort messages (source, destination, protocol, etc). The same parser you use to extract useful information from your Snort logs will work with Sagan syslog data! It just "works". No new parsing or data extraction techniques are needed. This idea applies to any SIEM technologies (ELSA, Logstash, etc). The final step is to get these Sagan log messages from your local system to your SIEM. In order to do this, we need the local syslog daemon to forward these events.

If your system uses syslog-ng as a logging daemon, you would want to add something like this to your syslog-ng configuration:

filter f_sagan { program("sagan*"); }; destination f_sagan_siem { udp("10.10.10.10" port 514); }; log { source(src); filter(f_sagan); destination(f_sagan_siem); };

If your system uses rsyslog as a logging daemon, you would want to add something like this to your rsyslog configurations.

If $programname == 'sagan*' then @10.10.10.10:514

For a older, more traditional syslog daemon, you would use something like this:

auth.alert @10.10.10.10

(Note: "10.10.10.10" would be your SIEM. After these changes are made, your syslog daemon will likely need to be reset or restarted).

This will allows Sagan to directly send alerts via syslog. I should note that if you use Barnyard2 with Sagan, you've always had this ability! One of the output formats Barnyard2 has is syslog! In fact, if you are using Barnyard2 with Sagan, you'll likely want to enable the syslog output in your Barnyard2 configurations! To configure with Barnyard2, you would add this to your configuration:

output alert_syslog: host=10.10.10.10:514, LOG_AUTH LOG_ALERT

With this sort of setup, Sagan can now share it's threat intelligence directly with your SIEM.

19.5 Sagan Flowbit

Posted by Kat Casey on June 08, 2015 These insights were provided by the expertise of Rob Nunley.

(Update: November 17th, 2018 - The term 'flowbit' is really tied to 'xbit')

Daniel Kahneman is a Doctor of Psychology who was awarded the Nobel Prize in Economic Sciences in 2002 (http://www.princeton.edu/~kahneman/). It may seem strange, initially, that a Psychologist would win one of the most world-renowned economics awards, but Dr. Kahneman's contributions can be applied to many fields; this includes cybersecurity. Dr. Kahneman's primary contribution was related to "human judgment and decision-making under uncertainty" (http://www.nobelprize.org/nobel_prizes/economic-sciences/laureates/2002/kahneman-facts.html), of which he has performed a great deal of research and experimentation.

Dr. Kahneman, who often performed experiments with Dr. Amos Tversky, may be best known for his research into System 1 and System 2 thinking. System 1 thinking uses heuristics, or quick and dirty "rules", to make instant and subconscious decisions. System 2 thinking involves logical and conscious thought to make decisions. Heuristics are susceptible to a number of systematic errors and pitfalls, but heuristics serve a purpose. (http://people.hss.caltech.edu/~camerer/Ec101/JudgementUncertainty.pdf).

Just as with everyday life, heuristics are often sufficient for many tasks and that utility extends to the realm network monitoring via NIDS and log analyses. Log messages contain information detailing the occurrence of an event. At best, a log message might indicate the source and destination of an action, the user(s) involved in the action, the catalyst of an action, and the outcome of an action. At worst, a log message may contain any one, or none, of those items. Sagan uses two primary methods for alert detection: heuristics (i.e., "rules" or signatures) and processors (e.g., reputational lookup of IP addresses). Some conclusions derived from the application of heuristics are valid—"login failure for user root from src IP 1.2.3.4" is pretty straightforward. A single log message is not always a valid indicator of an event, however, as we will explore below:

**4722: A user account was enabled.

Subject: Security ID: ACMEadministrator Account Name: administrator Account Domain: ACME Logon ID: 0x20bad

Target Account: Security ID: ACMEHumpty.Dumpty Account Name: Humpty-Dumpty Account Domain: ACME**

The above log message clearly states that "A user account was enabled", so what is the confusion? The log message, by itself, is missing context. If a Windows account is disabled and re-enabled, only the above log message will appear. If an account is created, however, there are always two messages created: 4720: A user account was created and 4722: A user account was enabled.

A Sagan feature developed specifically for the clustering of indicators in order to apply context to heuristics-based detection is flowbit. Flowbit, while not true System 2 thinking, empowers Sagan with the ability to trigger alerts for specific events only in the presence or absence of other events. Flowbits are given unique names based on what they are being used for (e.g., "created_enabled"). Sagan can measure for the presence or absence of events with flowbit by using a "flag" to represent whether or not a flowbit is set. An example of Sagan applying context for more informed decision-making can be observed by revisiting the Windows user account enabled example.

If a Windows account is disabled and re-enabled, there is no "account re-enabled" event. Instead, research was required to identify indicators and to find unique indicators which could be used for diagnosticity. As mentioned previously, creation of a new Windows account generates log messages for both enabled and created, but re-enabled accounts only generate account enabled events. The signatures below are used to determine when a Windows account has been re-enabled.

**alert syslog $EXTERNAL_NET any -> $HOME_NET any (msg:"[WINDOWS-AUTH] User account created [FLOWBIT SET]"; content: " 4720: "; program: Security*; classtype: successful-user; flowbits: set, created_enabled, 30; flowbits: noalert; reference: url,wiki.quadrantsec.com/bin/view/Main/5001880; sid: 5001880; rev:3;)

alert syslog $EXTERNAL_NET any -> $HOME_NET any (msg:"[WINDOWS-AUTH] User account re-enabled"; content: " 4722: "; content:! "$" ;program: Security*; flowbits: isnotset, by_src, created_enabled; classtype: successful-user; reference: url,wiki.quadrantsec.com/bin/view/Main/5001881; sid: 5001881; rev:3;)**

The first flowbit field in the first signature (sid: 5001880) notifies Sagan by using the flowbit command "set", provides a unique name for the flowbit, and declares how long the flowbit should remain active.

flowbits: set, created_enabled, 30

The flowbit details are stored in memory along with other information such as IP addresses involved. The second flowbit field in the first signature instructs Sagan not to produce an alert if this rule is triggered.

flowbits: noalert

The second signature contains only a single flowbit field, but this is what determines if an alert will trigger. This signature instructs Sagan that, if all other criteria for the signature match, check the flowbit table for a flowbit named created_enabled where the source IP address matches the newly identified source IP address (by_src). If the flowbit does not exist (isnotset), generate an alert stating that a user account has been re-enabled.

flowbits: isnotset, by_src, created_enabled

If there is still some confusion, we can examine once again why we are looking for a flowbit that does not exist in this scenario.

User account created created message && enabled message

User account re-enabled enabled message

If it is our intention to know when an account has been re-enabled, we do not want to trigger on any account enabled messages following an user account created message for the same source IP address. Context is provided by the presence or absence of the user account created message combined with the IP address being tracked.

Flowbit consists of three basic functions:

flowbits: set, (flowbit name), (expire time);

Instructs Sagan to create an entry in memory for the unique flowbit name for the duration, in seconds, given as an expire time.

flowbits: (unset|isset|isnotset), (by_src|by_dst|both|reverse|none), (flowbit name);"

Instructs Sagan how to respond to an alert with respect to a flowbit that has been set for a unique name. Possible actions are checking if the flowbit is set or is not set, as well as unsetting the flowbit if it exists. Search criteria is defined by tracking the source IP address, destination IP address, both IP addresses, the inverse of the original source and destination (i.e., source becomes destination / destination becomes source), or no tracking criteria.

flowbits: noalert;

This instructs Sagan not to generate an alert when a rule triggers, and is best used with initial indicators in a chain. Although flowbit does not have many features by itself, its power comes by chaining, or clustering, events in a multitude of combinations. Consider the following scenarios:

A Windows server shuts down normally, so logs are generated for each process that is killed. If a message stating that anti-virus software has been killed is observed in conjunction with a message stating that a server is shutting down, then that is expected. If a message stating that anti-virus software has been killed is observed but the server is not being shut down or restarted, then that is something that may be of interest to administrators and security analysts.

A user logging in to a system is normal. Observing five-thousand login failures followed by a login success may be suspect.

What if we want to track more than two indicators in succession? Sagan can handle that, too! Not only can Sagan chain numerous indicators, but an initial indicator in a chain can be used by multiple secondary indicators. Also, since Sagan can process whatever logs are sent to it, we can leverage Snort IDS logs to combine network events with system events.

Consider the following scenarios:

Snort logs (forwarded to Sagan) indicate a remote file inclusion attempt. This sets the RFI flowbit.

The attack, which was successful, causes the web server to request a Perlbot file. Sagan checks the RFI flowbit and, because the flowbit was set for the web server's IP address, we can receive an alert notifying us that there was a successful RFI attack.

If we have another "flowbits: set" instruction in our "flowbits: isset" signature, we have the ability to extend our chain. With reliable, valid indicators, we are able to receive increasingly relevant information with each additional signature. Let's extend the above scenario a little farther.

Snort logs (forwarded to Sagan) indicate a remote file inclusion attempt. This sets the RFI flowbit.

The attack, which was successful, causes the web server to request a Perlbot file. Sagan checks the RFI flowbit and, because the flowbit was set for the web server's IP address, we can receive an alert notifying us that there was a successful RFI attack. In addition to the alert, we set another flowbit called RFI_Download.

The web server runs a new process (detected via OSSEC, auditd, or some other service). Since the RFI_Download flowbit is set, we know that the new process started by our web server may be of interest to incident responders, so Sagan can send us another alert!

We'll discuss more advanced Sagan flowbit usage in a later blog post, but I hope that the example scenarios shown have at least opened the reader's mind to the possibilities the power and potential of flowbit.

All forms of heuristics are prone to various limitations and shortcomings, but flowbit helps overcome some of the potential pitfalls inherent in heuristics-based detection. Sagan's flowbit can increase accuracy and reduce false positives by requiring multiple indicators, potentially from multiple sources, before triggering an alert. Flowbit can be used to support incident responders, as shown above, by tracking indicators in real-time (this can also help with postmortem incident analysis). Flowbit also ensures that events occur within the context in which they are relevant.

The possibilities are limited by creativity, observability of events, and diagnosticity of indicators.

CHAPTER 20

Articles about Sagan

20.1 Reading

"Logging and processing logs from Windows 7. Timber!" - Linux Magazine - http://www.linux-magazine.com/content/download/61671/482426/version/1/file/072-073_kurt.pdf

"Analyzing Bro Logs with Sagan" (2015/09/10) - https://blog.zeek.org//2015/09/analyzing-bro-logs-with-sagan.html

"Sagan as a Log Normalizer" (2013/11/16) - https://isc.sans.edu/forums/diary/Sagan+as+a+Log+Normalizer/17039

"ELSA with Sagan" (2013/01/31) - http://blog.infosecmatters.net/2013/01/elsa-with-sagan.html

"Infoworld Sagan BOSSIE award" (2012/09/18) - https://www.infoworld.com/article/2606792/open-source-software/bossie-awards-2012--the-best-open-source-networking-and-security-software.html#slide17

20.2 Audio/Video

Champ discusses Sagan "Pauldotcom Security Weekly" (2013/12/12) - http://traffic.libsyn.com/pauldotcom/PaulDotCom-356-Part1.mp3

"Taking a bite out of logs with Sagan" at "Hackers On Planet Earth" (HOPE9) (2012/07) - https://www.youtube.com/watch?v=pMlAmteCjQo

Champ talks with the Jacksonville Linux User group about Sagan - https://www.youtube.com/watch?v=rySjNnEpjbI

20.3 Presentations/Papers

"Securing your Mikrotik Network" by Andrew Thrift (Presentation) - http://sagan.io/pdf/2_andrew.pdf

"Building wireless IDS systems using open source" - 2013? - http://sagan.quadrantsec.com/papers/wireless-ids/

"Defending the Homeland: Logging and Monitoring at home" by @nullthreat - http://sagan.io/pdf/BlackLodgeNSMOverview-Nullthreat.pdf

"Centralized and structured log file analysis with Open Source and Free Software tools" Bachelor Thesis by Jens Kühnel. - http://sagan.io/pdf/bachelor.pdf

CHAPTER 21

Getting help

The primary Sagan site is located at:

https://sagan.io

Sagan Github page is located at:

https://github.com/beave/sagan

If you are having issues getting Sagan to work, consider posting in the Sagan mailing list. This list is good for general configuration, install and usage questions.

https://groups.google.com/forum/#!forum/sagan-users

If you need to report a compile or programming issue, please use our Github.com issues page. That is located at:

https://github.com/beave/sagan/issues

If you want to chat about Sagan you can hit up our Discord channel!

https://discord.gg/VS6jTjH4gW

CHAPTER 22

TODO

- Documentation on new JSON decoders. (did json-input)
- Better documentation on syslog-ng, rsyslog and nxlog setup (pipe and JSON)
- external now powered by json

Index