License Server Extension

Release 202308.03

Sentieon, Inc

Oct 23, 2024

Contents

1 Introduction

This document describes how to extend the license server to provide additional authentication functionality. If you have any additional questions, please contact the technical support at Sentieon® Inc. at support@sentieon.com.

2 How the license server extension works

The license server extension adds another layer of user-defined authentication on top of the license server. It allows the user to define a custom authentication agent to be launched by the license server whenever a new license request comes in. Here is how the extension works.

1. On the license server side, the user needs to provide a script to be launched with the license server. The script can be written in any scripting language, for example, Python or shell. In this document, we will assume the script is a Python script named auth.py.
2. Launch the license server with option –auth pointing to the authentication script.

```
sentieon licsrvr --start --auth /path/to/auth.py /path/to/licfile
```

3. On the computing node, set two environment variables SENTIEON_AUTH_MECH and SENTIEON_AUTH_DATA. Through these environment variables, user can set any preset content or information gathered locally for authentication or other purposes.

```
export SENTIEON_AUTH_MECH=$SECURITY_MECHANISM

export SENTIEON_AUTH_DATA=$SECURITY_CONTEXT
```

4. Start a new Sentieon® job. When the job starts, the content of `$SENTIEON_AUTH_MECH` and `$SENTIEON_AUTH_DATA`, together with other local environment data will be sent to the license server in JSON format with the following keys:
   - mech: `$SENTIEON_AUTH_MECH`.
   - data: `$SENTIEON_AUTH_DATA`.
   - addr: Local IP address.
   - user: Current user.
   - groups: User groups current user belongs to.
5. On the license server, the above JSON string passed from the computing node will be piped to `auth.py` script through standard input. The script will parse and authenticate the JSON data. If the authentication is successful, the script will return with exit status 0. Otherwise, it will return with a non-zero exit status.
6. If `auth.py` returns non-zero, the license server will return license validation failure status to the requesting job. In that case, the job will quit with a “No license” error message.

3 License server extension script

3.1 Specification

The license server extension script needs to meet the following:

1. The script file’s permission is executable by the license server launcher.
2. The script accepts the input JSON string through standard input.
3. When authentication is successful, the script returns 0. Otherwise, it returns non-zero.

3.2 Example

On the license server, the license server is started with the following command:

```
sentieon licsrvr --start --auth /path/to/auth_unix.py lic_file.lic
```

An example of `auth_unix.py` file is below:

```
#!/usr/bin/env python
import json
import sys
import urllib

secret_key = “My_scret_key”

def main(argv):
    inputs = json.load(sys.stdin)
    print >>sys.stderr, inputs
    if inputs.get('mech') != 'unix':
```

(continues on next page)

(continued from previous page)

```
        print >>sys.stderr, "mech not unix"
        return -1
    if inputs.get('data') != secret_key:
        print >>sys.stderr, "Incorrect key"
        return -1
    if not isinstance(inputs.get('groups'), list):
        return -1
    if 'Developers' not in inputs.get('groups'):
        print >>sys.stderr, "Developers not in groups"
        return -1
    if 'baduser' == inputs.get('user'):
        print >>sys.stderr, "baduser is running the test"

    return 0

if __name__ == '__main__':
    sys.exit(main(sys.argv))
```

On the computing node, the following environment variable is set.

```
export SENTIEON_LICENSE=mylicsrvr:8992
export SENTIEON_AUTH_MECH=METHOD
export SENTIEON_AUTH_DATA=ASK_ME
```

Launch a job the computing node will pass the following JSON string to the license server:

```
{u'mech': u'METHOD', u'data': u'ASK_ME', u'addr': u'10.1.2.3',\
 u'groups': [u'Domain Users', u'Developers'], u'user': u'goodguy'}
```

Since 'METHOD' is not 'unix' as expected by the `auth_unix.py` script, `auth_unix.py` script will exit -1. The license server will print out the following output, and the requesting job will exit with a "No license" error message.

```
mech not unix
```

4 Cloud Examples

4.1 Google Computing Platform - Running a license server in a persistent e2-micro instance

In cloud environments, the most straightforward way to set up a license server is to have a very small continually running virtual machine acting as the license server. This setup will allow the license server to serve licenses to any other VM within the license server's Subnet.

The following instructions will guide you through setting up a e2-micro instance as a license server:

1. Choose an GCP Region to work in. Choose (or create) the Subnet where you will run the Sentieon® tools.
2. Note the CIRD block for your Subnet.
3. Create a firewall rule set for the license server (Fig. 4.1 and Fig. 4.2). The rules must:

   a. Allow inbound TCP communication at a specific port (we use 8990 by default as it is not typically used by other applications). This port is used to allow communication between the compute nodes and the license server. You should open TCP at the desired port across your Subnet's CIDR block (10.128.0.0/20 in Fig. 4.1) to whitelist traffic from within your Subnet.

   b. Allow outbound HTTPS communication to Sentieon® license master at master.sentieon.com (IP 52.89.132.242). This is necessary to enable license validation and updates.

   3. Allow SSH communication. This is necessary for administration.

   d. (Recommended) Allow ICMP communication. This allows PMTU with minimal security risk.

4. Launch a e2-micro instance to run the license server. Be sure to launch the instance in the region you selected earlier and connect to the network interface with the updated firewall rules created in step 3.
5. Send your Sentieon® support representative the Private IP address of your instance, together with the TCP port you opened in step 3.a.
6. Download the Sentieon® tools and your license file to the instance.
7. Start the license server with the following command.

```
sentieon licsrvr --start [-l <licsrvr_log>] <license_file>
```

8. (Optional) Confirm the license server is working correctly and serving licenses to the license server instance, by running following commands; the second command will return the number of available licenses.

```
sentieon licclnt ping -s <IP_addresss>:<PORT> || (echo "Ping Failed"; exit 1)
sentieon licclnt query -s <IP_address>:<PORT> klib
```

9. Create a firewall rule set for the compute nodes (Fig. 4.3 and Fig. 4.4). The rules should:

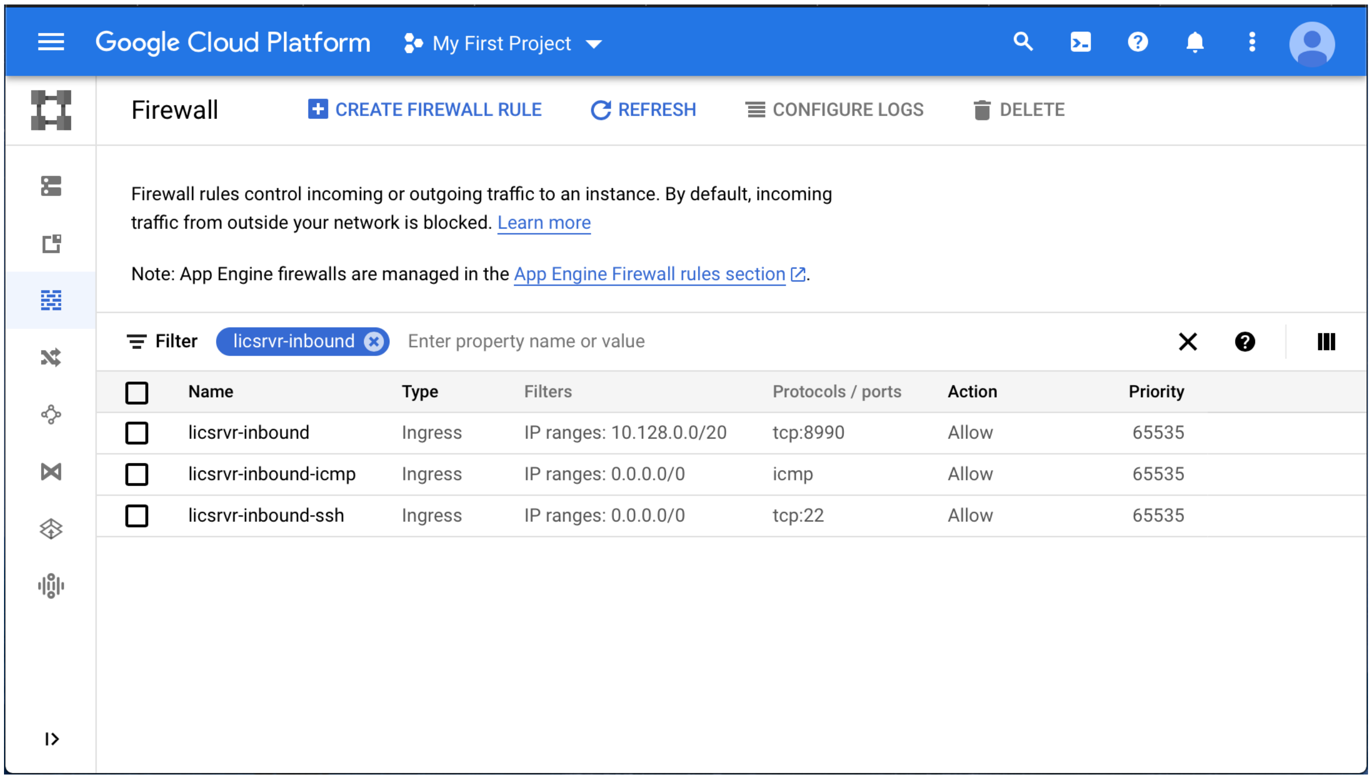
   1. Allow inbound SSH communication for administration.

   b. Allow outbound TCP communication at the port chosen in step 3-a, which will be open across your Subnet's CIDR block.

   c. (Recommended) Allow ICMP communication to facilitate communication between the license server and compute nodes.

10. Launch another e2-micro instance within the same Subnet to test the license server. Be sure launch the instance in the region you selected earlier and connect to the network interface with the updated firewall rules created in step 9.
11. Confirm the license server is working and serving licenses within the Subnet by running the following commands; the second command will return the number of available licenses.

```
sentieon licclnt ping -s <IP_addresss>:<PORT> || (echo "Ping Failed"; exit 1)
sentieon licclnt query -s <IP_address>:<PORT> klib
```



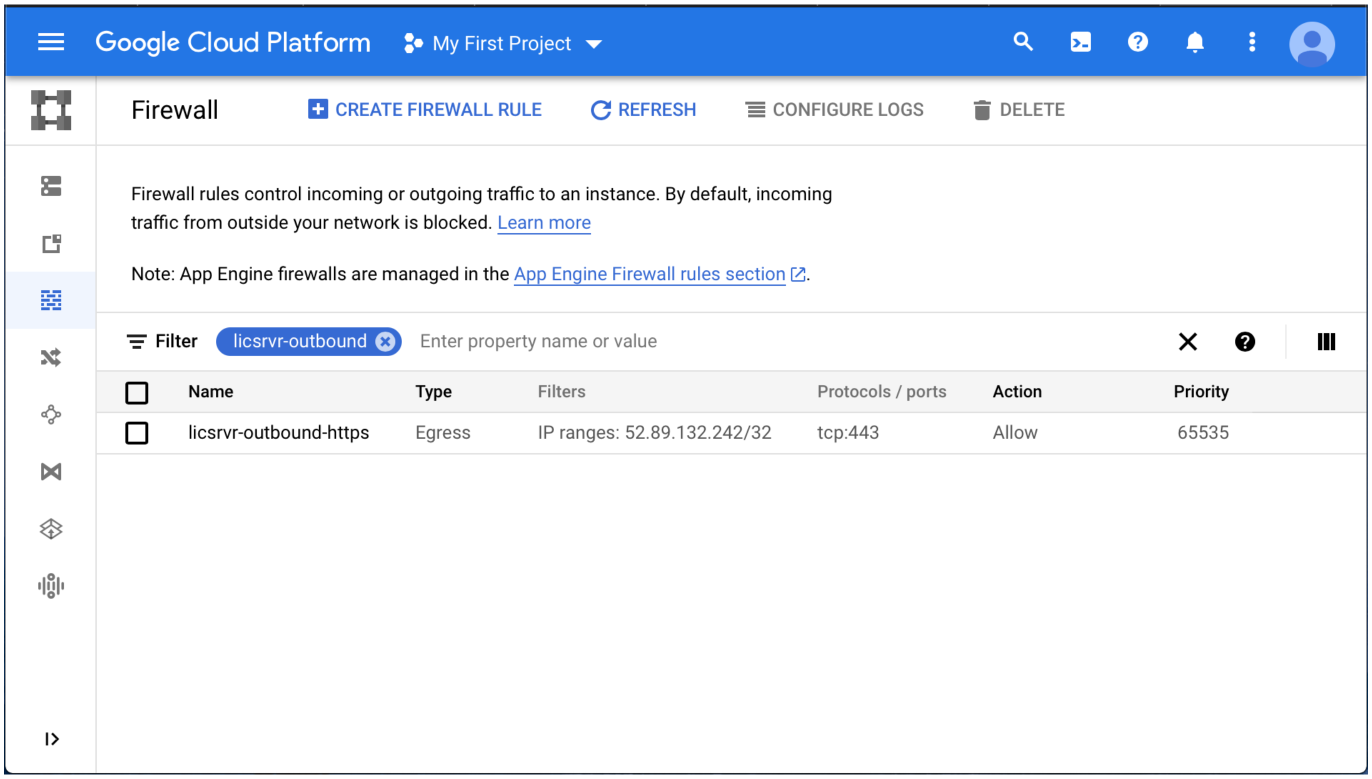
Firewall rules control incoming or outgoing traffic to an instance. By default, incoming traffic from outside your network is blocked. Learn more

Note: App Engine firewalls are managed in the App Engine Firewall rules section.

Filter licsrvr-inbound Enter property name or value

Name	Type	Filters	Protocols / ports	Action	Priority
licsrvr-inbound	Ingress	IP ranges: 10.128.0.0/20	tcp:8990	Allow	65535
licsrvr-inbound-icmp	Ingress	IP ranges: 0.0.0.0/0	icmp	Allow	65535
licsrvr-inbound-ssh	Ingress	IP ranges: 0.0.0.0/0	tcp:22	Allow	65535

Fig. 4.1: Example license server inbound firewall rules



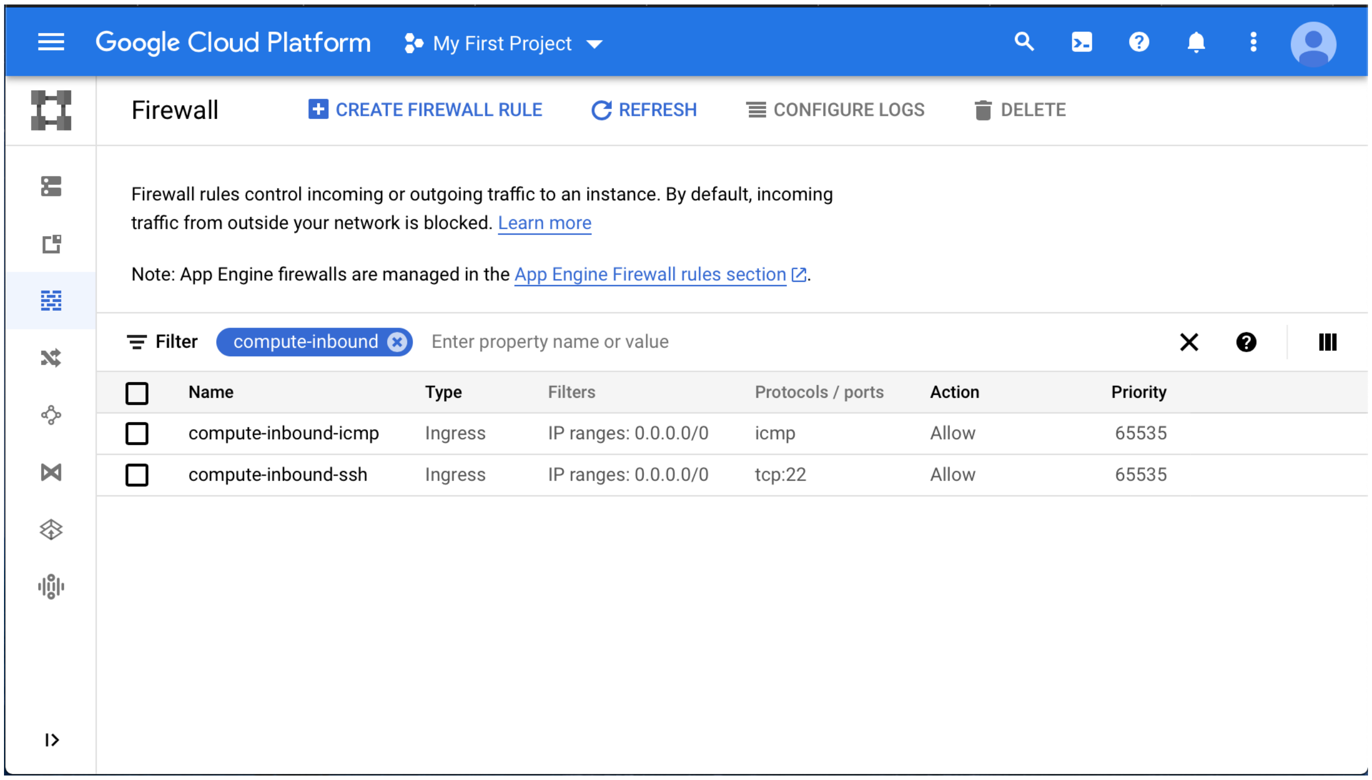
Firewall rules control incoming or outgoing traffic to an instance. By default, incoming traffic from outside your network is blocked. Learn more

Note: App Engine firewalls are managed in the App Engine Firewall rules section.

Filter licsrvr-outbound Enter property name or value

Name	Type	Filters	Protocols / ports	Action	Priority
licsrvr-outbound-https	Egress	IP ranges: 52.89.132.242/32	tcp:443	Allow	65535

Fig. 4.2: Example license server outbound firewall rules

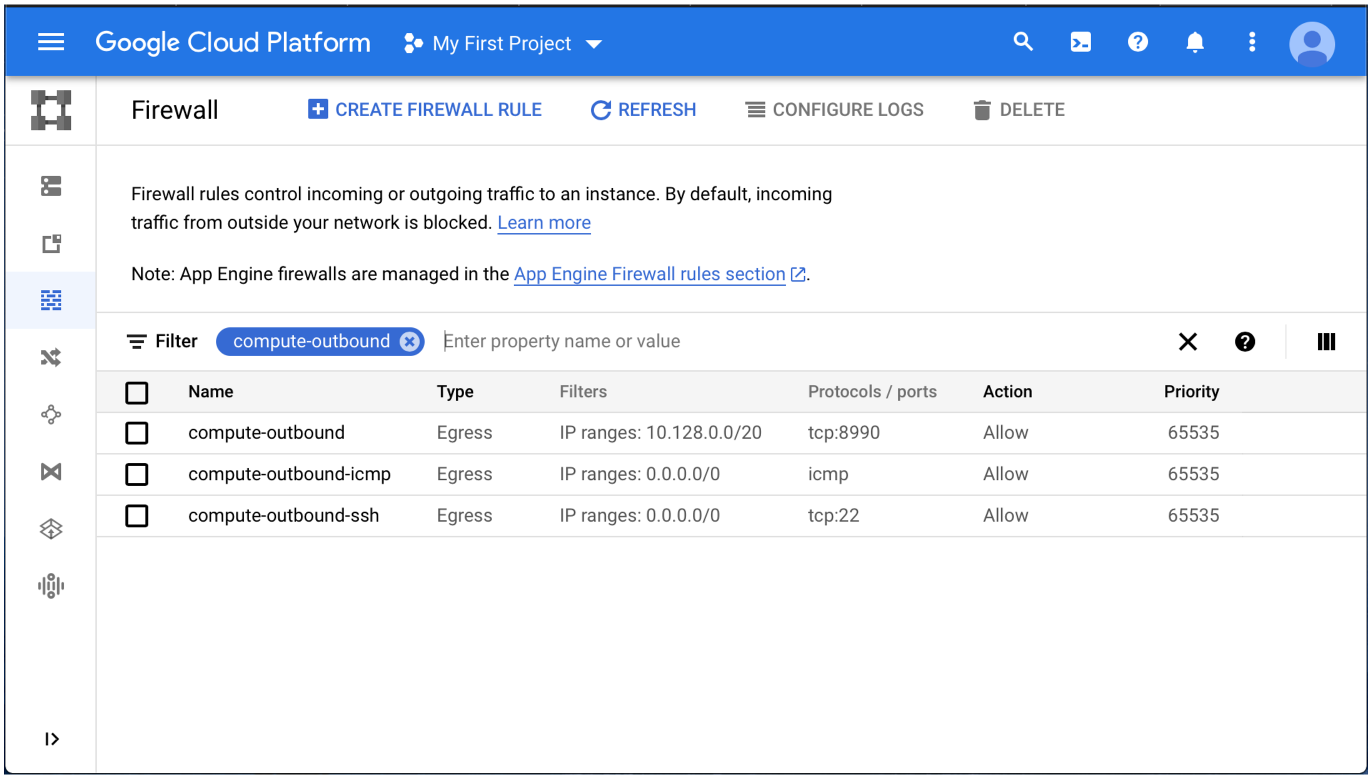


Firewall rules control incoming or outgoing traffic to an instance. By default, incoming traffic from outside your network is blocked. Learn more

Note: App Engine firewalls are managed in the App Engine Firewall rules section.

Name	Type	Filters	Protocols / ports	Action	Priority
compute-inbound-icmp	Ingress	IP ranges: 0.0.0.0/0	icmp	Allow	65535
compute-inbound-ssh	Ingress	IP ranges: 0.0.0.0/0	tcp:22	Allow	65535

Fig. 4.3: Example compute nodes inbound firewall rule



Firewall rules control incoming or outgoing traffic to an instance. By default, incoming traffic from outside your network is blocked. Learn more

Note: App Engine firewalls are managed in the App Engine Firewall rules section.

Name	Type	Filters	Protocols / ports	Action	Priority
compute-outbound	Egress	IP ranges: 10.128.0.0/20	tcp:8990	Allow	65535
compute-outbound-icmp	Egress	IP ranges: 0.0.0.0/0	icmp	Allow	65535
compute-outbound-ssh	Egress	IP ranges: 0.0.0.0/0	tcp:22	Allow	65535

Fig. 4.4: Example compute nodes outbound firewall rules