Getting started

This guide provides all the basic information you need to start using the Wazuh API.

Starting and stopping the API

The API starts at boot time. To control or check the wazuh-api service, use the systemctl or service command.

Systemd systems

```
systemctl start/status/stop/restart wazuh-api
```

SysVinit systems

```
service wazuh-api start/status/stop/restart
```

Hello world!

In order to check if everything is working as expected, you can use cURL to do a *request*:

```
$ curl -u foo:bar -k https://127.0.0.1:55000?pretty
{
   "error": 0,
   "data": "Welcome to Wazuh HIDS API"
}
```

Explanation:

- `curl`: This is a command-line tool for sending requests and commands over HTTP and HTTPS.
- `-u foo:bar`: Specify a username and password to authenticate with the API.
- `-k`: Allow connections to SSL sites with self signed certs.
- `https://127.0.0.1:55000`: This is the API URL to use if you are running the command on the manager itself.
- `?pretty`: This parameter makes the JSON output more human-readable.

Basic concepts

Basic concepts about making API requests and understanding their responses:

- The *base URL* for each request is `https://IP:55000/` or `http://IP:55000/`, depending on if you enabled and set up SSL in the API.
- All responses are in *JSON format* with the following structure:

Field	Description
error	0 if everything was fine and an error code otherwise.
data	data requested. Only if error is equal to 0.
message	error description. Only if error is different to 0.

- Example response without errors:

```
{ "error": "0", "data": "Welcome to Wazuh HIDS API" }
```

- Example response with errors:

```
{ "error": "603", "message": "The requested URL was not found on this server" }
```

- Responses containing collections of data will return a maximum of 500 elements. You should use the *offset* and *limit* parameters to iterate through large collections.
- All responses have an HTTP status code: 2xx (success), 4xx (client error), 5xx (server error), etc.
- All requests accept the parameter *pretty* to convert the JSON response to a more human-readable format.
- The API log is stored on the manager as `/var/ossec/logs/api.log`.

Use cases

This section will present several use cases to give you a taste for the API's potential. You can find details about all possible API requests in the reference section.

Exploring the ruleset

Often when an alert fires, it is helpful to know details about the rule itself. The following request enumerates the attributes of rule *1002*:

```
curl -u foo:bar -k "https://127.0.0.1:55000/rules/1002?pretty"
```

```
{
  "error": 0,
  "data": {
    "totalItems": 1,
    "items": [
      {
        "status": "enabled",
        "pci": [],
        "description": "Unknown problem somewhere in the system.",
        "file": "syslog_rules.xml",
        "level": 2,
        "groups": [
          "syslog",
          "errors"
        ],
        "id": 1002,
        "details": {
          "options": "alert_by_email",
          "match": "$BAD_WORDS"
        }
      }
    ]
  }
}
```

It can also be helpful to know what rules are available that match specific criteria. For example, we can show all rules with a group of **web**, a PCI tag of **10.6.1**, and containing the word **failures**, which returns only one rule in this case.

```
curl -u foo:bar -k "https://127.0.0.1:55000/rules?group=web&pci=10.6.1&search=failures&pretty"
```

```
{
  "error": 0,
  "data": {
    "totalItems": 1,
    "items": [
      {
        "status": "enabled",
        "pci": [
          "10.6.1",
          "10.2.4",
          "10.2.5",
          "11.4"
        ],
        "description": "Multiple web authentication failures.",
        "file": "nginx_rules.xml",
        "level": 10,
        "groups": [
          "authentication_failures",
          "nginx",
          "web"
        ],
        "id": 31316,
        "details": {
          "same_source_ip": null,
          "frequency": "6",
          "if_matched_sid": "31315",
          "timeframe": "240"
        }
      }
    ]
  }
}
```

Mining the file integrity monitoring database of an agent

You can use the API to show information about all the files monitored by syscheck. For example, you can enumerate all monitored files on agent *000* (the manager) with extension *.py* that have been modified. In order to be concise, "*limit=1*" has been used in this example to limit the results to a single record.

```
curl -u foo:bar -k "https://127.0.0.1:55000/syscheck/000/files?offset=0&limit=1&event=modified&search=.py&pretty"
```

```
{
  "error": 0,
  "data": {
    "totalItems": 1,
    "items": [
      {
        "uid": 0,
        "scanDate": "2016-07-14 10:58:45",
        "user": "root",
        "file": "/home/example.py",
        "modificationDate": "2016-07-14 10:58:18",
        "octalMode": "100777",
        "inode": 270323,
        "event": "modified",
        "size": 8,
        "sha1": "a38c98822f783fd45c256fe8fc928300c169d138",
        "group": "root",
        "gid": 0,
        "permissions": "-rwxrwxrwx",
        "md5": "b7f912e271b6c3e86ba2787f227d984c"
      }
    ]
  }
}
```

In case you need to find a file using its md5/sha1 hash, you can do so with a simple request like this:

```
curl -u foo:bar -k "https://127.0.0.1:55000/syscheck/000/files?hash=9d0ac660826f4245f3444b0247755c7229f1f9fe&pretty"
```

```
{
  "error": 0,
  "data": {
    "totalItems": 1,
    "items": [
      {
        "uid": 0,
        "scanDate": "2016-07-14 08:49:27",
        "user": "root",
        "file": "/etc/default/cron",
        "modificationDate": "2014-10-25 22:04:09",
        "octalMode": "100644",
        "inode": 262805,
        "event": "added",
        "size": 955,
        "sha1": "9d0ac660826f4245f3444b0247755c7229f1f9fe",
        "group": "root",
        "gid": 0,
        "permissions": "-rw-r--r--",
        "md5": "eae0d979b5007d2af41540d8c2631359"
      }
    ]
  }
}
```

Listing outstanding rootcheck issues

Rootcheck requests are very similar to the syscheck ones. In order to get all rootcheck issues with an *outstanding* status you can run this request:

```
curl -u foo:bar -k "https://127.0.0.1:55000/rootcheck/000?status=outstanding&offset=0&limit=1&pretty"
```

```
{
  "error": 0,
  "data": {
    "totalItems": 3,
    "items": [
      {
        "status": "outstanding",
        "oldDay": "2016-07-14 08:49:28",
        "readDay": "2016-07-14 08:49:28",
        "event": "System Audit: SSH Hardening - 1: Port 22 {PCI_DSS: 2.2.4}. File: /etc/ssh/sshd_config"
      }
    ]
  }
}
```

Starting the manager and dumping its configuration

It is possible to use the API to interact with the Manager in many ways. For example, you can stop/start/restart it or get its state with this simple request:

```
curl -u foo:bar -k -X PUT "https://127.0.0.1:55000/manager/restart?pretty"
```

```
{
  "error": 0,
  "data": [
    {
      "status": "running",
      "daemon": "wazuh-moduled"
    },
    {
      "status": "running",
      "daemon": "ossec-maild"
    },
    {
      "status": "running",
      "daemon": "ossec-execd"
    },
    {
      "status": "running",
      "daemon": "ossec-analysisd"
    },
    {
      "status": "running",
      "daemon": "ossec-logcollector"
    },
    {
      "status": "running",
      "daemon": "ossec-remoted"
    },
    {
      "status": "running",
      "daemon": "ossec-syscheckd"
    },
    {
      "status": "running",
      "daemon": "ossec-monitord"
    }
  ]
}
```

You can even dump the manager’s current configuration with the below request (response shortened for brevity):

```
curl -u foo:bar -k "https://127.0.0.1:55000/manager/configuration?pretty"
```

```
{
  "error": 0,
  "data": {
    "global": {
      "email_notification": "no",
      "white_list": [
        "127.0.0.1",
        "^localhost.localdomain$",
        "10.0.0.2"
      ],
      "jsonout_output": "yes",
      "logall": "yes"
    },
    "...": {"...": "..."}
  }
}
```

Playing with agents

Of course we can work with agents. This enumerates **active** agents:

```
curl -u foo:bar -k "https://127.0.0.1:55000/agents?offset=0&limit=1&status=active&pretty"
```

```
{
  "error": 0,
  "data": {
    "totalItems": 1,
    "items": [
      {
        "status": "Active",
        "ip": "127.0.0.1",
        "id": "000",
        "name": "LinMV"
      }
    ]
  }
}
```

Adding an agent is now easier than ever. Just send a request with the agent name and its IP.

```
curl -u foo:bar -k -X POST -d '{"name":"NewHost","ip":"10.0.0.8"}' -H 'Content-Type:application/json' "https://127.0.0.1:55000/agents?pretty
```

```
{
  "error": 0,
  "data": "019"
}
```

You can fetch an agent’s key like this:

```
curl -u foo:bar -k "https://127.0.0.1:55000/agents/019/key?pretty"
```

```
{
  "error": 0,
  "data":
"MDE5IGFkZmFmZGFkZmFkZmFkZmEgMTg1LjE2LjIxMS44OCBjN2Y2YzFhMjc4NWI1NjBhOWZiZGJiNjY2ODMwMzdlODNkMjQwNDc5NmUxMDI2
Yzk1ZTBmMmY2MDQ5ZDU1Mjlj"
}
```

Conclusion

We hope you now better appreciate the potential of the Wazuh API. Remember to check out the reference document to discover all the available API requests. A nice summary can also be found here: summary.