Hashtopussy Communication Protocol

seinlc

December 27, 2016

Introduction

The communication between Hashtopussy Clients and Server is always in JSON formatted values. When sending a request to the server, it should be a POST with value "query=[JSON]".

Errors

In case of an error with the query which the client sends to the server, the response will have following format with the corresponding action which was requested and the error message which should help in getting information about this error.

```
{
  "action":"register",
  "response":"ERROR",
  "message":"Provided voucher does not exist."
}
```

Commands

Register

When registering a new client to the server, following query should be used: (Important: To make sure an agent will be recognized as GPU agent, you have to get some info in the 'gpus' field. The server searches for keywords like 'ati', 'nvidia' etc.)

```
{
  "action":"register",
  "voucher":"89GD78tf",
  "uid":"230-34-345-345",
  "name":"client name",
  "os":0,
  "gpus":[
    "ATI HD7970",
    "ATI HD7970"
  ]
}
```

All values are required and must not be empty. The operating system should be set as following: 0 => Linux, 1 => Windows, 2 => Mac OS X.
As response the server will send following on success:

```
{
  "action":"register",
  "response":"SUCCESS",
  "token":"GHhgdf(/&"
}
```

The token will be the one, the client should use for his following connections on this server.

Login

The client logs in to the server.

```
{
  "action":"login",
  "token":"GHhgdf(/&"
}
```

As response, the client will get back the agent timeout setting of the server (in seconds) if the login was successful.

```
{
  "action":"login",
  "response":"SUCCESS",
  "timeout":30
}
```

Update

This is used by the client to check if there is an update available for the client script.

```
{
  "action":"update",
  "version":"0.1.0 ALPHA"
}
```

The server responds either with

```
{
  "action":"update",
  "response":"SUCCESS",
  "version":"OK"
}
```

if the client has the newest version. Or he sends

```
{
  "action":"update",
  "response":"SUCCESS",
  "version":"NEW",
  "data":"........"
}
```

with 'data' containing the new version of the script.

Download

This command is used to either download the 7z binary to extract Hashcat, or to get information where to download the newest Hashcat version and some additional informations about it.

```
{
  "action":"download",
  "type":"7zr",
  "token":"GHhgdf(/&"
}
```

This will result in a non-JSON response. The server will simply provide the corresponding 7z binary for the agent as download. To check if a new Hashcat binary is available, following command is used.

```
{
  "action":"download",
  "type":"hashcat",
  "token":"GHhgdf(/&",
  "version":"3.0.1",
  "force":1
}
```

'force' is optional and is be used when the server should send version information anyway, even if the client already has the newest version. 'version' contains the current version the client has.
The server will respond with following information:

```
{
  "action":"download",
  "response":"SUCCESS",
  "version":"NEW",
  "url":"https://hashcat.net/files/hashcat-3.01.7z",
  "files":[
    "hashcat.hctune",
    "hashcat64.bin"
  ],
  "rootdir":"hashcat-3.0.1",
  "executable":"hashcat64.bin"
}
```

Error

In case there happens an error with Hashcat on the client, he can submit the error message to the server where it will be assigned to the agent and shown on the page.

```
{
  "action":"error",
  "task":45,
  "token":"GHhgdf(/&",
  "message":"This is a sample error message!"
}
```

The server will confirm the error messages:

```
{
  "action":"error",
  "response":"SUCCESS"
}
```

Get File

If the client needs a file for running a task (Wordlist or Rule File) he needs to request the url for downloading it.

```
{
  "action":"file",
  "token":"GHhgdf(/&",
  "task":56,
  "filename":"rockyou.txt"
}
```

The server sends then the url, where the client can download the file from.

```
{
  "action":"file",
  "response":"SUCCESS",
  "url":"get.php?file=5&token=GHhgdf(/&"
}
```

Get Hashlist

The client can download a Hashlist/Superhashlist.

```
{
  "action":"hashes",
  "token":"GHhgdf(/&",
  "hashlist":23
}
```

If there occurs any error with this request, the server will respond in JSON format with error information. If everything is correct, it will send a list of hashes which were requested.

The server then will send the whole hashlist as plain-text download on success, otherwise he will send an error.

Get Task

The client requests the current task he should work on.

```
{
  "action":"task",
  "token":"GHhgdf(/&"
}
```

The server will send then a task to do:

```
{
  "action":"task",
  "response":"SUCCESS",
  "task": 25,
  "wait": 0,
  "attackcmd": "#HL# -a 3 ?a?a?a?a",
  "cmdpars": "--hash-type=0",
  "hashlist": 13,
  "bench": "new",
  "statustimer": 5,
  "files": [
    "rockyou.txt",
    "wordlist.txt"
  ]
}
```

or send this if there is no task available:

```
{
  "action":"task",
  "response":"SUCCESS",
  "task":"NONE"
}
```

Get Chunk

The client requests the current task he should work on.

```
{
  "action":"chunk",
  "token":"GHhgdf(/&",
  "taskId": 5
}
```

The server will send then a chunk to work on:

```
{
  "action":"chunk",
  "response":"SUCCESS",
  "chunk": 13,
  "skip": 45000,
  "length": 10000
}
```

or he can send one of the following answers which should be done then:
In case it is required to measure the keyspace of the corresponding task.

```
{
  "action":"chunk",
  "response":"SUCCESS",
  "chunk":"keyspace_required"
}
```

In case the current task is already fully dispatched and the agent should request for a new task.

```
{
  "action":"chunk",
  "response":"SUCCESS",
  "chunk":"fully_dispatched"
}
```

In case there is no benchmark for this task on the current agent. So it should start benchmarking then.

```
{
  "action":"chunk",
  "response":"SUCCESS",
  "chunk":"benchmark"
}
```

Send Keyspace

The client sends the calculated keyspace of a task.

```
{
  "action":"keyspace",
  "token":"GHhgdf(/&",
  "taskId":35,
  "keyspace":568000
}
```

The server should reply following on success:

```
{
  "action":"keyspace",
  "response":"SUCCESS",
  "keyspace": "OK"
}
```

Send Benchmark

The client sends the tested benchmark of a task. Benchmark type can be 'old' or 'new'.

```
{
  "action":"bench",
  "token":"GHhgdf(/&",
  "taskId":35,
  "type": "old",
  "result": "674.43"
}
```

The server should reply following on success:

```
{
  "action":"bench",
  "response":"SUCCESS",
  "benchmark": "OK"
}
```

Send Solve

In the given interval of the task, the client is working on, it should report the progress in following way:

```
{
  "action":"solve",
  "token":"GHhgdf(/&",
  "chunk":35,
  "keyspaceProgress":568000,
  "progress":500,
  "total":600,
```

```
  "speed":570000,
  "state": 3,
  "cracks": [
    "7fde65673fd28736423f23423786f:thisisplain",
    "7fde65673f28987f7423f2342378f:otherplain",
  ]
}
```

The server will reply with following:

```
{
  "action":"solve",
  "response":"SUCCESS",
  "cracked": 2,
  "skipped": 0
}
```

If there were cracks on the same hashlist from some other clients or view the web-interface, the server will send the hashes which should be zapped from the hashlist

```
{
  "action":"solve",
  "response":"SUCCESS",
  "cracked": 2,
  "skipped": 0,
  "zap": [
    "87634876fe7f7df5e76f56a757",
    "7863487563874ffefe67575745"
  ]
}
```