Hashtopolis Communication Protocol (V2)

s3in!c ⟨s3inlc@hashes.org⟩

September 7, 2018

Introduction

The communication between Hashtopolis Clients and Server is always in JSON formatted values. When sending a request to the server, it should be a POST containing the JSON data.

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

Connection Check

To test the connection to the server or to test if an URL is valid, you can issue a test action where the server just should respond with success.

```
{
  "action":"testConnection"
}
```

Response:

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

Register

When registering a new client to the server, following query should be used:

```
{
  "action":"register",
  "voucher":"89GD78tf",
  "name":"client name"
}
```

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.

Send Client Information

To send information about the client operating system and the available hardware. (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.)

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.
This command should be issued on client startup to update if any hardware changes occurred since the last connect.

```
{
  "action":"updateInformation",
  "token":"GHhgdf(/&",
```

```
  "uid":"230-34-345-345",
  "os":0,
  "devices":[
    "ATI HD7970",
    "ATI HD7970"
  ]
}
```

The response says if the update was successful.

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

Login

The client logs in to the server.

```
{
  "action":"login",
  "clientSignature":"generic-python",
  "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
}
```

Check Client Version

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

```
{
  "action":"checkClientVersion",
  "version":"0.43.5",
  "type":"csharp",
  "token":"GHhgdf(/&"
}
```

The server responds either with

```
{
  "action":"checkClientVersion",
```

```
  "response":"SUCCESS",
  "version":"OK"
}
```

if the client has the newest version. Or he sends

```
{
  "action":"update",
  "response":"SUCCESS",
  "version":"NEW",
  "url":"https://example.com/getclient.php"
}
```

with 'url' providing the download url for the new version.

Download Binary

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

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

The server then will send a url where to downooad the binary:

```
{
  "action":"downloadBinary",
  "response":"SUCCESS",
  "url":"https://example.com/getapp.php"
}
```

The PRINCE download link must be configured on the server, then the client can retrieve the url.

```
{
  "action":"downloadBinary",
  "type":"prince",
  "token":"GHhgdf(/&"
}
```

The server then will send a url where to download the binary:

```
{
  "action":"downloadBinary",
  "response":"SUCCESS",
  "url":"https://github.com/hashcat/princeprocessor/releases/download/v0.22/princeprocessor-0.22.7z"
}
```

To download the Hashcat/Cracker binary, following command is used.

```
{
  "action":"downloadBinary",
  "type":"cracker",
  "binaryVersionId": 45
  "token":"GHhgdf(/&"
}
```

'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":"downloadBinary",
  "response":"SUCCESS",
  "name":"Hashcat",
  "url":"https://hashcat.net/files/hashcat-3.01.7z",
  "executable":"hashcat64.bin"
}
```

Client Error

In case there happens an error with Hashcat/Cracker 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":"clientError",
  "taskId":45,
  "token":"GHhgdf(/&",
  "message":"This is a sample error message!"
}
```

The server will confirm the error messages:

```
{
  "action":"clientError",
  "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":"getFile",
  "token":"GHhgdf(/&",
  "taskId":56,
  "file":"rockyou.txt"
}
```

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

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

Get Hashlist

The client can download a Hashlist/Superhashlist.

```
{
  "action":"getHashlist",
  "token":"GHhgdf(/&",
  "hashlistId":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 the url for the raw download of the hashes.

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

where 'url' contains the download link for the raw data.

Get Task

The client requests the current task he should work on.

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

The server will send then a task to do (benchType can be 'speed' or 'run'):

```
{
  "action":"getTask",
  "response":"SUCCESS",
  "taskId": 25,
  "attackcmd": "#HL# -a 3 ?a?a?a?a --hash-type=0",
  "hashlistId": 13,
  "bench": 30,
```

```
  "statustimer": 5,
  "benchType":"speed",
  "crackerId":45
  "hashlistAlias":"#HL#",
  "files": [
    "rockyou.txt",
    "wordlist.txt"
  ],
  "keyspace":0,
  "usePrince": false,
  "enforcePipe": false
}
```

or send this if there is no task available:

```
{
  "action":"getTask",
  "response":"SUCCESS",
  "taskId": null,
  "reason": "No task available!"
}
```

Get Chunk

The client requests the current task he should work on.

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

The server will send then a chunk to work on:

```
{
  "action":"getChunk",
  "response":"SUCCESS",
  "status":"OK",
  "chunkId": 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":"getChunk",
  "response":"SUCCESS",
```

```
  "status":"keyspace_required"
}
```

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

```
{
  "action":"getChunk",
  "response":"SUCCESS",
  "status":"fully_dispatched"
}
```

In case there is an update of the cracker binary available, the server sends this:

```
{
  "action":"getChunk",
  "response":"SUCCESS",
  "status":"cracker_update"
}
```

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

```
{
  "action":"getChunk",
  "response":"SUCCESS",
  "status":"benchmark"
}
```

In case there was a health check requested on the server, it will notify the client.

```
{
  "action":"getChunk",
  "response":"SUCCESS",
  "status":"health_check"
}
```

Send Keyspace

The client sends the calculated keyspace of a task.

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

The server should reply following on success:

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

Send Benchmark

The client sends the tested benchmark of a task. Benchmark type can be 'speed' or 'run'. If the 'speed' benchmark type is used, there are two values to be sent separated by ':'.
The 'speed' benchmark type is using the –progress-only switch of hashcat.
The 'run' benchmark type uses the old method of Hashtopolis to benchmark a task in running the given task for some time.

```
{
  "action":"sendBenchmark",
  "token":"GHhgdf(/&",
  "taskId":35,
  "type": "run",
  "result": "674"
}

{
  "action":"sendBenchmark",
  "token":"GHhgdf(/&",
  "taskId":35,
  "type": "speed",
  "result": "674:674.74"
}
```

The server should reply following on success:

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

Send Progress

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

```
{
  "action":"sendProgress",
  "token":"GHhgdf(/&",
  "chunkId":35,
  "keyspaceProgress":568000,
  "relativeProgress":"4545"
  "speed":570000,
  "state": 3,
  "cracks": [
    [
      "7fde65673fd28736423f23423786f",
      "thisisplain",
      "746869736973706c61696e",
```

```
      "787523889"
    ],
    [
      "7fde65673f28987f7423f2342378f",
      "otherplain",
      "6f74686572706c61696e",
      "78652"
    ]
  ],
  "gpuTemp": [
    67
  ],
  "gpuUtil": [
    99
  ]
}
```

RelativeProgress is the progress of the current chunk between 0 and 10'000 (in percent * 100). The cracks are sent as array resulting from hashcat's output format 15.

The server will reply with following:

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

Get deleted files

The client can request a list of deleted filenames from the server to be able to clean up unused files.

```
{
  "action":"getFileStatus",
```

```
  "token":"GHhgdf(/&",
}
```

The server will then provide a list of filenames:

```
{
  "action": "getFileStatus",
  "response": "SUCCESS",
  "filenames": [
    "top10000.txt"
  ]
}
```

Get health check

When the client got notified that the server wants to do a health check, it should request the data.

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

The server will provide the task to run and the hashes to use for it.

```
{
  "action": "getHealthCheck",
  "response": "SUCCESS",
  "attack": "#HL# -a 3 -1 ?l?u?d ?1?1?1?1?1",
  "crackerBinaryId": 1,
  "hashes": [
    "9f650d4ef70eead895e7a03ce0f83e88",
    "24a7bdfd14bca7f9d14154c0800a833d"
  ],
  "checkId": 7,
  "hashlistAlias": "#HL#"
}
```

Send health check

When the client executed a health check it should send back the results.

```
{
  "action": "sendHealthCheck",
  "token": "GHhgdf(/&",
  "numCracked": 23,
  "start": 1536306884,
  "end": 1536306902,
  "numGpus": 1,
  "errors": [
```

```
    "nvmlDeviceGetTemperatureThreshold(): Not Supported",
    "nvmlDeviceGetTemperatureThreshold(): Not Supported"
  ],
  "checkId": 7
}
```

On success the server will just reply with OK.

```
{
  "action": "sendHealthCheck",
  "response": "OK"
}
```