Zombie and C&C Network Protocol Documentation

Lucas Williamson, Glahens Paul, TJ, Selam Van Voorhis

`

Figure 1.1 Zombie/ C&C Protocol Initialization Diagram

Figure 1.2 RUN Request Diagram

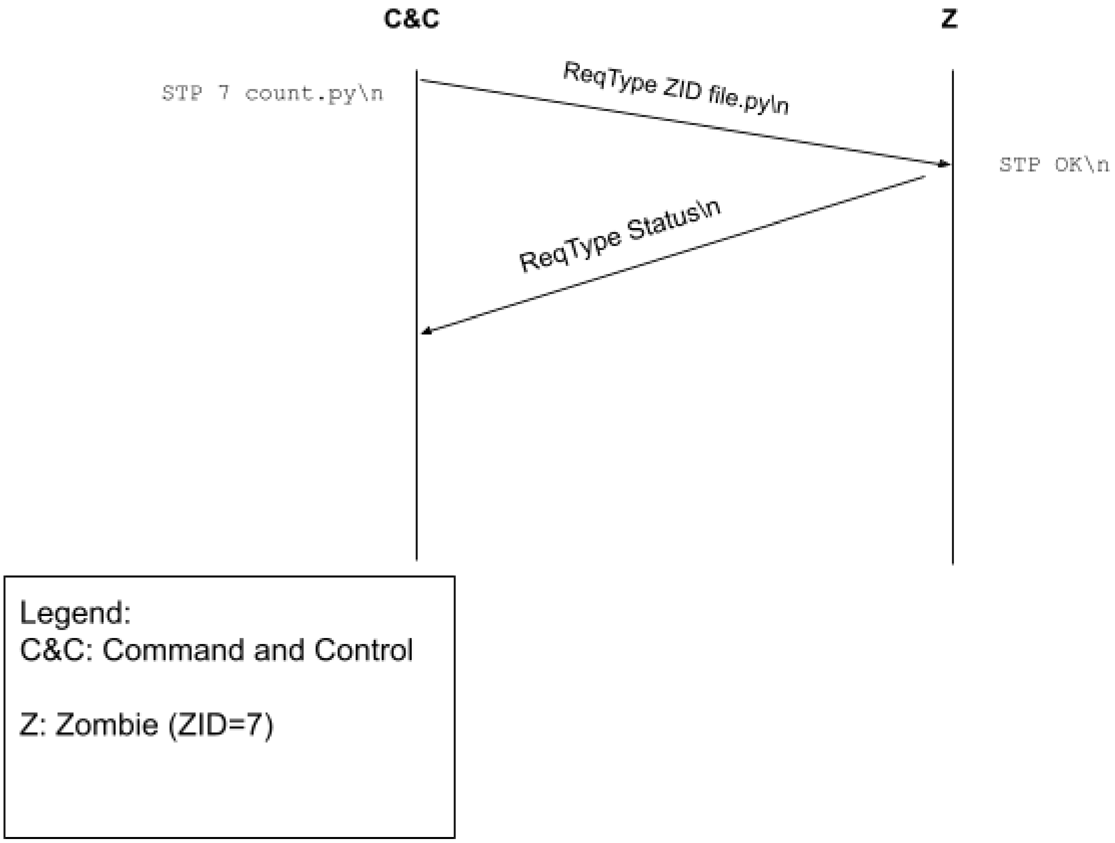

Figure 1.3 STOP Request Diagram

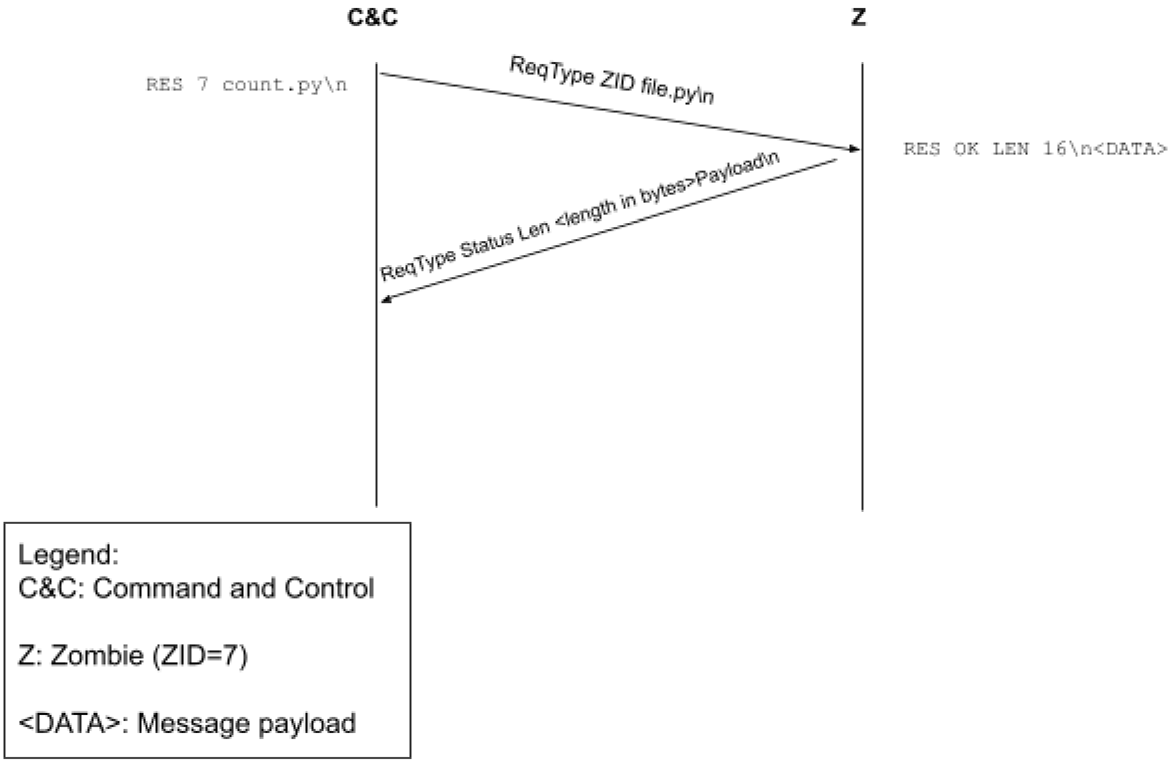

Figure 1.4 RESULT Request Diagram

Introduction

For this protocol the most important considerations we took into account included clear, fast and reliable data request and response messages, time efficient implementation and ensuring server control of the clients.

Initialization Protocol:

Messages between command and control and zombies are sent via TCP connection. Upon startup, the command and control process opens a TCP server socket at a predetermined port–TCP is used for reliability and to maintain a connection with a zombie process for later commands. In order to establish an initial connection between a newly-executed zombie process and the command and control process, the zombie sends an initialization code message to the command and control process, which will be received through a newly-opened connection socket. The initialization code used in the first message is `AWAKE`. It is recommended that the command and control generates a zombie ID integer (ZID) before storing the ZID and socket in an appropriate data structure (hashtable-like structures are recommended–e.g. a Python dictionary). The command and control process responds to the zombie process with an initialization status ~~(typically `OK`)~~ and the assigned ZID. The zombie process then sends an additional initialization code and the received ZID; this second initialization code is `READY`. At this point, the zombie is prepared for requests. An example of the initialization protocol is shown in Figure 1.1.

Protocol Messages

The command and control starts the communication for action requests on a zombie after the initialization protocol. It can make one of three requests: run a python script, stop a running python script, or get the results from a python script held by a zombie (whether it be the output or a message saying that the script is running). The command and control can ask for specific zombies to execute a request or can ask several zombies to complete a request at a time. The zombies always send back an acknowledgement back to the server so that if an action cannot be fulfilled the server can send another request or update changes about a zombie.

When the server is asking for a zombie to run a script, the request will be made up of the RequestType `RUN`, ZID, and the name of the script. The corresponding zombie will send a message back with a Status response consisting of the RequestType and a Status indicating if the run was successful. The Status `OK` indicates successful completion while the Status `NF` indicates a running python script matching the requested filename was not found. An example of a Run exchange is shown in Figure 1.2.

The C&C can make other requests, such as stopping a running file. In order to do this the C&C sends a message consisting of the request type `STO`, ZID, and the script name. When the C&C asks a zombie to stop running a script, the zombie will reply with a response consisting of the RequestType and a Status indicating if the stop was successful. The Status `OK` indicates successful completion while the Status `NF` indicates a running python script matching the requested filename was not found. The Status `NT` indicates that the script failed to terminate. An example of a Run exchange is shown in Figure 1.3.

Another request the C&C can make is a request for the result of the file that it told the zombie to run. This result can be the output of the program or a message status indicating to the C&C that the program is still running. The request made by the C&C contains the RequestType `RES`, the ZID, and the script name. When the zombie receives this message, it will send a response consisting of the RequestType, Status indicating if the stop was successful, the `LEN` keyword followed by the length of the payload in bytes, a newline character, and the payload. The Status `OK` indicates successful completion while the Status `NFIN` indicates that the requested process has not returned results yet. The Status `NF` indicates that a python script matching the provided filename was not found. An example of a Run exchange is shown in Figure 1.4.

Example Messages:

Initialization Protocol

Request from Zombie: InitCode: **AWAKE**\n

```
AWAKE\n
```

Response from C&C: InitStatus: **OK/ ERROR- NO** ZID: **1**\n

```
OK 1\n

NO\n
```

Response from Zombie: InitCode: **READY** ZID: **1**\n

```
READY 1\n
```

Run script request from the server

Request From C&C: ReqType: **RUN** ZID: **1** ScriptName: **count.py**\n

```
RUN 1 count.py\n
```

Reply From Zombie: ReqType: **RUN** Status: **OK/ ERROR- NFNN**\n

```
RUN OK\n

RUN NF\n
```

Stop request sent from the server

Request From C&C: ReqType: **STP** ZID: **1** ScriptName: **count.py**\n

```
STP 1 count.py\n
```

Reply From Zombie: ReqType: **STP** Status: **OK/ ERROR- NF/NT**\n

```
STP OK\n

STP NF\n

STP NT\n
```

Result request sent from the server

Request From C&C: ReqType: **RES** ZID: **1** ScriptName: **count.py** \n

```
    RES 1 count.py\n
```

Reply From Zombie: Status: **PAYLOAD/ ERROR- NF/NFIN**

```
RES OK LEN 300\n<payload>

RES NF\n

RES NFIN\n
```

Conversation

When we started planning we chose to have the C&C send only one request per script at a time, this meant that a TCP communication link would only be alive for one script request, in the implementation only one request per zombie per script is made. We later decided to keep the connection. For this protocol we implemented two important things for HTTP. Like HTTP request messages, our request messages specified the action first, whereas HTTP has two types of request messages (GET and HEAD), we have three (RUN, STP and RES). We also decided to use TCP as our transport layer protocol like HTTP. [redacted]

Consider cutting the ZID