

# Monitor Server v2.5

Coleman Kane

March 2, 2004

## Contents

|          |  |          |
|----------|--|----------|
| <b>1</b> | <b>Communication Modes with the Monitor</b>            | <b>2</b> |
| 1.1      | Active Connections . . . . .                           | 2        |
| 1.2      | Passive Connections . . . . .                          | 2        |
| <b>2</b> | <b>Communication Protocol</b>                          | <b>2</b> |
| 2.1      | Monitor Directives . . . . .                           | 3        |
| 2.1.1    | RESULT: Codes . . . . .                                | 3        |
| 2.1.2    | REQUIRE: Codes . . . . .                               | 3        |
| 2.1.3    | WAITING: Codes . . . . .                               | 4        |
| 2.1.4    | COMMENT: Codes . . . . .                               | 4        |
| 2.1.5    | COMMAND_ERROR: Codes . . . . .                         | 4        |
| 2.2      | Client Commands . . . . .                              | 4        |
| 2.2.1    | Logging In to the Monitor for the First Time . . . . . | 5        |
| 2.2.2    | The PASSWORD Command . . . . .                         | 5        |
| 2.2.3    | The HOST_PORT Command . . . . .                        | 5        |
| 2.2.4    | The ALIVE Command . . . . .                            | 6        |
| 2.2.5    | The QUIT Command . . . . .                             | 6        |
| 2.3      | The Command Mode . . . . .                             | 6        |
| 2.3.1    | The HELP Command . . . . .                             | 7        |
| 2.3.2    | The MAKE_CERTIFICATE Command . . . . .                 | 7        |
| <b>3</b> | <b>Advanced Features</b>                               | <b>7</b> |
| 3.1      | Connection Encryption . . . . .                        | 7        |
| 3.1.1    | Unauthenticated Connections . . . . .                  | 7        |
| 3.1.2    | Authenticated Connections . . . . .                    | 8        |

### Abstract

This document will describe the protocol and design used by client hosts in interacting on the Monitor server. The Monitor is a server provided to implement a game for a set of participants to play. The purpose of the game is to earn as many points as possible through the use of various interfaces and features. The participants may implement various mechanisms for attacks against other participants to prevent victims from receiving points and even for stealing points directly from other participants.

## 1 Communication Modes with the Monitor

The Monitor will provide multiple services for interfacing from the client machines. There are two connection methods: Active and Passive. The *Active Connection* is established when a game participant initiates a connection to the Monitor. The *Passive Connection* is established when the monitor initiates a connection to a listening client.

### 1.1 Active Connections

The monitor provides active connections to the game participant by listening on a pre-selected port. The selected port(s) for the project is 8080 of helios.ececs.uc.edu. Active connections require participants to connect to the selected port and authenticate themselves to the listening monitor via a pre-selected cookie or password. A participant logs in over an active connection.

### 1.2 Passive Connections

The login protocol, described below, requires participants to start a *passive server* which listens for Monitor messages on a legal port of their choice. The port number is given to the Monitor during the login handshake. The passive server serves two functions. First, it is used to accumulate wealth for the participant. The Monitor connects to a participant's passive server from time to time to verify that the server is up and running. Acknowledgement of so-called *alive queries* is necessary for the participant to receive wealth from the Monitor. Alive queries may happen at any time, for any reason. Second, passive connections are the primary medium for any sort of out-of-band connections, such as for proxied user-to-user communications.

## 2 Communication Protocol

All communication between active client, passive client, and Monitor is in the form of line-buffered, text messages. Each text message may contain one or more lines of text, called a *message group*. Each line of a message group is called a *Directive*. The format of each directive in a message group is the following:

DIRECTIVE: Message Body

Possible directives are given below. The client normally responds to the message group with a single line of text with the following format:

COMMAND ARG1 ARG2 ARG3

The delimiter of command and arguments is the blank character. Possible commands are given below.

A communication sequence is a series of exchanges between Monitor and client to achieve some result. It begins with the Monitor sending a message group specifying a command that the client is expected to issue next. The client issues the command with arguments. The Monitor responds with another message group and so on until some result is achieved.

## 2.1 Monitor Directives

The types of directives used by the Monitor fall into the following categories: **RESULT:**, **REQUIRE:**, **ERROR:**, **COMMENT:**, and **WAITING:**. This list may be amended in future revisions of the monitor (the current version is 2.9). Directive format is given above. The body of a directive specifies associated detail which informs the client of the status at the current point of the communication sequence. Message groups always end with the **WAITING:** directive. A common communication sequence looks something like this:

```
...
CLIENT> ALIVE S54DFSESX4234DX
MONITOR> RESULT: ALIVE Identity verified.
MONITOR> REQUIRE: HOST_PORT
MONITOR> WAITING:
CLIENT> HOST_PORT LOCALHOST 31337
...
```

One might think of the **WAITING:** directive as a flow-control message from the Monitor. Upon receiving the **WAITING:** directive, the client knows that the monitor is expecting it to act on the provided information and issue a command when ready. The following table explains the function of each directive.

| Name                  | Description  |
|-----------------------|--|
| <b>REQUIRE:</b>       | Tells the client what command or course of action the monitor is requesting.   |
| <b>RESULT:</b>        | Tells the client the result of performing an action. This is thought of as the return value of the client's command.                                       |
| <b>COMMENT:</b>       | Peripheral information to the client regarding the monitor, the current connection, or both.   |
| <b>WAITING:</b>       | Tells the client that the Monitor is waiting for input.  |
| <b>COMMAND_ERROR:</b> | Tells the client that an error occurred. The state of the current command is undefined. The client should take some course of action to resolve the issue. |

### 2.1.1 RESULT: Codes

When the monitor responds to the action(s) of one of the clients, it sends a **RESULT** directive to the client. This directive has the following format:

```
RESULT: Operation Successful
```

The above directive is the result of some operation the client performed. The body of the directive may not necessarily be all caps or all lowercase and may consist of many blank or comma separated tokens. The blanks are not considered separators in the same way as they are in the client commands. After a `<tt>RESULT:</tt>` is received, a client may parse the body to determine what course of action to take next.

### 2.1.2 REQUIRE: Codes

The Monitor uses the **REQUIRE:** directive to demand information from a client. This directive requests that the client perform a specific action. Typically, the action is invocation of a command which has been determined by the Monitor and which is included in the **REQUIRE:** directive. In the sample communication sequence given above, the Monitor issues a **REQUIRE: HOST\_PORT** directive and expects the client to respond with a **HOST\_PORT** command. After receiving the **WAITING:** directive, and starting a passive

server on port 31337 of the localhost, the client sends the `HOST_PORT LOCALHOST 31337` command to the Monitor. More information on commands is given below.

`REQUIRE: HOST_PORT`

### 2.1.3 WAITING: Codes

This directive has no body. Its presence merely signifies it is waiting for the client to some specified information to the Monitor.

### 2.1.4 COMMENT: Codes

The Monitor may place comments anywhere in a message group before the `WAITING:` directive. Comments are generally informational messages that may be of use to the client. They do not have a specific format, except that a comment message may follow the `COMMENT:` token. For example, a common message announcing the Monitor looks like this: `COMMENT: Monitor Server Version 2.9.`

### 2.1.5 COMMAND\_ERROR: Codes

`COMMAND_ERROR:` directives are sent by the Monitor in response to a failed command or connection. An error will result if a command does not succeed as expected. The connection remains alive and the client is expected to take some action to recover from the error. Most errors are designated with a `COMMAND_ERROR:` code to specify that they are a result of a failed command.

## 2.2 Client Commands

A client sends a command upon receiving the `WAITING:` directive from the server. A command consists of the command type followed by any arguments that might be necessary for the Monitor to execute the command successfully. Any command that a client sends without proper arguments will result in an error message directive giving a description of what arguments the command was supposed to be run with. Since the commands available to the user are numerous, only a few key commands will be listed here. The rest will be listed in the command reference towards the end of this document. The commands that will be covered by this section are outlined here:

| Command Name     | Usage   | Description  |
|------------------|---|--|
| IDENT            | IDENT <i>username</i> [ <i>key-for-encryption</i> ] | Identifies the connecting user to the Monitor. Also exchanges key parts for use by the encryption services.    |
| PASSWORD         | PASSWORD <i>password</i>                            | Give the monitor th client's password.   |
| ALIVE            | ALIVE <i>alive-cookie</i>                           | Give the monitor a shared session cookie for authentication  |
| HOST_PORT        | HOST_PORT <i>hostname tcpport</i>                   | Give the monitor the socket of your listen server, so that it may connect to your client later.                |
| MAKE_CERTIFICATE | MAKE_CERTIFICATE <i>exp modulus</i>                 | Give the monitor your public key for encryption so that it may register your key in its certificate authority. |
| QUIT             | QUIT  | Ends the current connection.   |

### 2.2.1 Logging In to the Monitor for the First Time

When you log into the Monitor for the first time your connection is not encrypted. You may already have a pre-selected username and password (set by the Monitor Administrator), or you may be creating a new participant account if the Monitor provides such a facility (available in practice tournaments). Either way, you should register yourself with the Monitor to maintain some good level of security for subsequent connections.

#### The IDENT Command

The IDENT command initiates login and is almost always the first command you will send to the monitor. No other commands are possible without first identifying yourself to the Monitor. When you first connect to the monitor you will receive a message group similar to the following.

```
COMMENT: Monitor Server Version 2.9
REQUIRE: IDENT
WAITING:
```

Upon receiving this, your client should identify itself to the Monitor. The easiest way to accomplish this is to send the monitor your username. For example:

```
IDENT PLAYER
```

### 2.2.2 The PASSWORD Command

After receiving the IDENT command, the Monitor requests a password from your active client as follows:

```
RESULT: IDENT PLAYER
REQUIRE: PASSWORD
WAITING:
```

If you are creating a new player, then this will set the stored password from this time forward. If you already have a registered password, then you will need to give that, or else the Monitor will give you the error message:

```
COMMAND_ERROR: Invalid player password.
```

The monitor takes this password and then generates a special cookie, which gets returned to the client. The client must save this cookie (to disk maybe), as it will be needed later to authenticate to the monitor again. This is known as a *session-key*. You will need to hold onto this key and keep it secret from others. The monitor will request this key from you in future communications.

Assuming the IDENT command was successful, the client responds to the WAITING: directive with a command such as the following:

```
PASSWORD MYLITTLESECRET
```

### 2.2.3 The HOST\_PORT Command

The server will respond to the PASSWORD command with:

```
RESULT: PASSWORD VKJHJ6FSK3JGM
REQUIRE: HOST_PORT
WAITING:
```

In this example, MYLITTLESECRET is the password the participant sends to the Monitor. The cookie chosen by the Monitor is VKJHJ6FSK3JGM. It must be remembered! At this stage the participant's passive server must be started and listening on a port which it chooses randomly. Suppose the port chosen is 31337 and the server is started on `rhodes.ececs.uc.edu`. Then the client sends the following command to the Monitor:

```
HOST_PORT RHODES.ECECS.UC.EDU 31337
```

The monitor may connect to this port from different source ports, and even (possibly) from different IP addresses. Upon connecting to the passive server, the Monitor exchanges two pieces of secret information. This allows a client to authenticate the Monitor, and the Monitor to authenticate the client. The Monitor takes the SHA-1 hash of the chosen password (not the session-key) and gives that to the passive server. The Monitor also requests from the passive server the session-key given previously by the result of the `PASSWORD` command. The checksum given by the Monitor is generated by taking the magnitude of the result of `MessageDigest.digest()` as the positive magnitude of an integer. The following shows the communication between the Monitor and the passive server:

Here is a demonstration of the passive connection:

```
PLAYER_PASSWORD_CHECKSUM: 224726e0160bf4a844f445424f2c81e03cf739cb
COMMENT: Monitor Server Version 2.9
REQUIRE: IDENT
WAITING:
IDENT PLAYER
RESULT: IDENT
REQUIRE: ALIVE
WAITING:
```

#### 2.2.4 The ALIVE Command

The passive server uses the checksum to authenticate the Monitor, then sends the session-key to the Monitor in an `ALIVE` command as follows:

```
ALIVE VKJHJ6FSK3JGM
RESULT: ALIVE Identity verified.
REQUIRE: QUIT
WAITING:
```

#### 2.2.5 The QUIT Command

The passive server responds with the `QUIT` command. This ends the passive session.

```
QUIT
```

This is the end of verification. The Monitor then sends a message group to the active client telling it whether it succeeded in authenticating a passive server. This looks as follows:

```
RESULT: HOST_PORT Command succeeded.
REQUIRE: COMMAND
WAITING:
```

The active client is now in command mode.

### 2.3 The Command Mode

Once authentication and login have completed, the active client is in Command Mode. At this stage, the participant is at the root of the command interface and has the freedom to commit any sort of actions that the Monitor accepts. There are numerous commands available to the participant, and they are covered later on. This section will focus on a few commands that are useful to the new user.

### 2.3.1 The HELP Command

Very important to the novice participant is the `HELP` command. This command provides a list of all of the commands that can be accepted by the Monitor.

### 2.3.2 The MAKE\_CERTIFICATE Command

The `MAKE_CERTIFICATE` command is used to register a public encryption key for use in RSA encryption facilities with the Monitor. The `MAKE_CERTIFICATE` command takes two arguments, the public encryption key exponent (commonly called  $e$ ) followed by the public key modulus (commonly called  $n$ ). The monitor does not perform any sort of checks to verify that the numbers are cryptographically strong. Such is the responsibility of the key generation method used. Each number is expected to be in base-32 numerical format (`BigInteger.toString(32)`), and they should be a positive number. The Monitor responds with a new number which is the monitor-signed hash of the supplied key. Here is a demonstration of the client registering a public key:

```
MAKE_CERTIFICATE 2001 342h23k3123ngf3h23jj3223432hng23fff32gg23j
RESULT: CERTIFICATE 02m21jhbb23jjm23jj23bbh32h
```

This is only an example - the values above are not actually generated values.

The Monitor handles the `MAKE_CERTIFICATE` command as follows. It creates two new `BigInteger` numbers out of each of the keys. It then converts the numbers to byte arrays using `BigInteger.toByteArray()`. It loads an instance of the SHA-1 Message Digest hashing algorithm and sends the exponent byte array to it, followed by the modulus byte array. It then gets a digest and runs its RSA decrypt function on the value. This has the effect of creating a new value that another person, having the Monitor's public encryption key, can verify that the Monitor is the one who made the certificate. The encrypt function of the Monitor's public key may be used to compare the hash with what has been received from another participant. As long as the client can trust the Monitor's public key, the identity is considered authenticated.

## 3 Advanced Features

By this point you should have a completely operational client that is up and running and receiving connections from the monitor.

### 3.1 Connection Encryption

The monitor supports connection-level encryption. This facility is provided to help secure the users against network sniffing attacks. The encryption is supported with the second (optional) argument to `IDENT`. If this argument is left blank, then the connection is plain text. There are two modes to the encryption, *Authenticated* and *Unauthenticated*.

#### 3.1.1 Unauthenticated Connections

Unauthenticated connections are encrypted connections where the connection may be encrypted, but the authenticity of the source is not verified. This is the case when a client connects to the monitor and has not yet been registered in the certification authority database. The client will need the public key of the monitor, which will be publicly available. Prior to the `IDENT` the client machine will generate a new 256-bit RSA keypair with an encryption exponent of 65537 (0x10001 in hex, 2001 in the canonical base32). The modulus of the generated public half is subsequently encrypted with the monitor's public key and sent over as the second argument to `IDENT`. This value is subsequently received and decrypted by the monitor. This is paired with the defined exponent value above to create the client's public encryption key. Next, the monitor generates a random 256-bit string and encrypts this value with the reconstructed public key from the client. The server then sends this back to the client in the result directive, and the client may decrypt this value. The following steps will be to construct the shared secret between the two

systems. Each side has the same pair of 256-bit strings (the public key modulus from the client, and the server-generated random string). These should be converted into byte arrays and will be assembled to create the shared secret. The shared key is build as follows: take the first byte (`serverHalf[0]`) from the server generated value, and put it in the new shared key array. Second, take the first byte from the client-generated value and place that into the shared key array. Keep alternating this pattern 256 times until you reach the end of the two key halves.

The following formula should demonstrate this relationship in a mathematical sense. The shared secret is  $k$ , the server half is  $s$  and the client half is  $c$ :

$$c_n = k_{2n+1}$$

$$s_n = k_{2n}$$

### 3.1.2 Authenticated Connections

Authenticated connections are sockets where the server can use the registered RSA key to verify the identity of the connecting user. This can be accomplished via the use of the *Certification Authority* within the monitor. When a game participant connects to the monitor, after registering a public key certificate via the `Make_CERTIFICATE` command, the monitor may now use this registered key for encrypting its shared secret half to be given to the connected participant. In this case, the value that the client sends using the `IDENT` command is no longer a short, 256-bit RSA key modulus, rather, it is simply a random 256-bit string, encrypted with the Monitor's public key. The monitor receives this value, and saves the decrypted version as the participant's half. It immediately generates its own random 256-bit string as above. Rather than using the value submitted by the client for the RSA encryption key, it already has a stored (and stronger) key in its database. It fetches this and encrypts the server-half with it to send with the `RESULT:` directive to the participant machine. If the participant is really who they said that they are, they will be able to decrypt this value and properly build the shared key. If not, they will receive gibberish and the connection will ultimately fail.