Minecraft Monitor

Lukas Smith - 1902745
Year 4 Ethical Hacking
CMP408 IoT and Cloud Secure Development
2022/23

Contents

Introduction

Importance of Project

The Minecraft Monitor is a downsized example of IoT Monitoring (Dynatrace, 2023), a vital section of IoT with many real-world applications. Companies such as DataDog provide IoT Monitoring on an enormous scale, allowing organisations to gauge performance based on specific parameters. Monitoring gives organisations the data they need to make informed decisions and troubleshoot effectively (Splunk, 2022). Additionally, IoT monitoring can ensure that all system logs are stored together, making it easier for engineers to rapidly identify software issues or security concerns (DataDog, 2023).

Minecraft Monitors' purpose is to monitor the player status of a Minecraft server using a Raspberry Pi 0 W, LEDs and an OLED screen to identify players leaving and joining. The player list is displayed on a webserver (EC2) alongside an offline player list provided by DynamoDB.

Security is a significant factor for this project, as several credentials must be used to communicate with the Minecraft Server, EC2 instance and DynamoDB. Ensuring these credentials are stored and used securely is vital, as mishandling them could allow them to be stolen and misused. As AWS is scalable and charges based on use, a substantial amount of money could be charged to the root AWS account if compromised.

Finally, as an LKM is developed and used in this project, the LKM must be handled properly to ensure that all GPIO pins are correctly released on the Raspberry Pi. For example, if the program were to open the LKM device and then crash without safety measures, the LKM would be left open, making it unremovable and leaving GPIO pins unreachable. If the GPIO pins were left in on on state, damage could be done.

Objectives

The objectives of Minecraft Monitor have been split into three main sections, Software, Hardware and Infrastructure.

Software

- Checks Minecraft Server for players
    - On player change, light up the corresponding LED using the LKM and display the username on the OLED screen for one second
- Current online players are to be displayed on OLED and webserver with the list of offline players on the webserver and the last time they were online

Hardware

- Two LEDs to be controlled by an LKM which is written to by a userspace program
- An OLED screen controlled by an external Python Library

Infrastructure

- An EC2 instance to host the webserver
- DynamoDB to store the list of offline players

Procedure

The procedure has been split into hardware, software and infrastructure components.

Hardware

First, the two LEDs and the OLED screen were wired to the Pi. To control the two LEDs, an LKM was developed using C and Visual Studio Code. The LKM uses a header file that includes a structure used for controlling the GPIO pin and the definitions for reading and writing to the GPIO (although reading was not necessary in this case, it was still implemented).

```
#ifndef LEDCONTROLLER_H
#define LEDCONTROLLER_H

#include <linux/ioctl.h>

typedef struct gpio_pin {
        unsigned int pin;
        int value;
} gpio_pin;

#define IOCTL_LEDCONTROLLER_GPIO_READ 0x65
#define IOCTL_LEDCONTROLLER_GPIO_WRITE 0x66

#define LED_ON 1
#define LED_OFF 0

#define  DEVICE_NAME "ledControllerDev"
#define  CLASS_NAME  "ledControllerCLS"

#endif
```

Figure 1 - ledController.h

On insertion, the LKM takes in a parameter called ‘led_gpios’, an array of integers declaring the GPIO pins. The initialisation function requests the listed GPIOs and creates the device `/dev/ledControllerDev`. Should the initialisation error or the module is unloaded, all requested GPIOs are turned off and safely released, and the device is removed.

The main function the LKM uses is ‘device_ioctl’, which is called using the userspace application, and takes in a command such as ‘IOCTL_LEDCONTROLLER_GPIO_WRITE’ to write to the LEDs using the structure defined in the header file.

The LKM was set up to load on the startup of the Raspberry Pi using `modprobe`, with GPIO pins 23 and 24.

Software

The main python script ‘main.py’ runs on the Raspberry Pi. The LED device driver is handled safely using a ‘try’ and ‘except’ clause, meaning that should the program end due to a Keyboard Interrupt or an unforeseen error, the driver will be safely released for another program to use.

The LKM was accessed using the ‘fcntl’ library, and another library called ‘ctypes’ was used to simulate a C-type Structure to communicate with the GPIO using the LKM. RCON was implemented using the ‘mctools’ library to query the Minecraft Server, SSH was implemented using the ‘paramiko’ library to connect to the EC2 instance and access to DynamoDB was implemented using the ‘boto3’ library. RCON, SSH and DynamoDB require credentials, these have been stored in an environment file.

Storing them in this file makes it easier to keep them secure when committing the code to a version control site such as Github, as the .env file can be added to .gitignore and will not be included in the repository. Additionally, the file locations are stored in the .env file, making it easier to keep them consistent should they change.

Once all the global variables have been declared, and the RCON, SSH and DynamoDB clients have all been set up, the main program loop starts.

The main program is as follows:

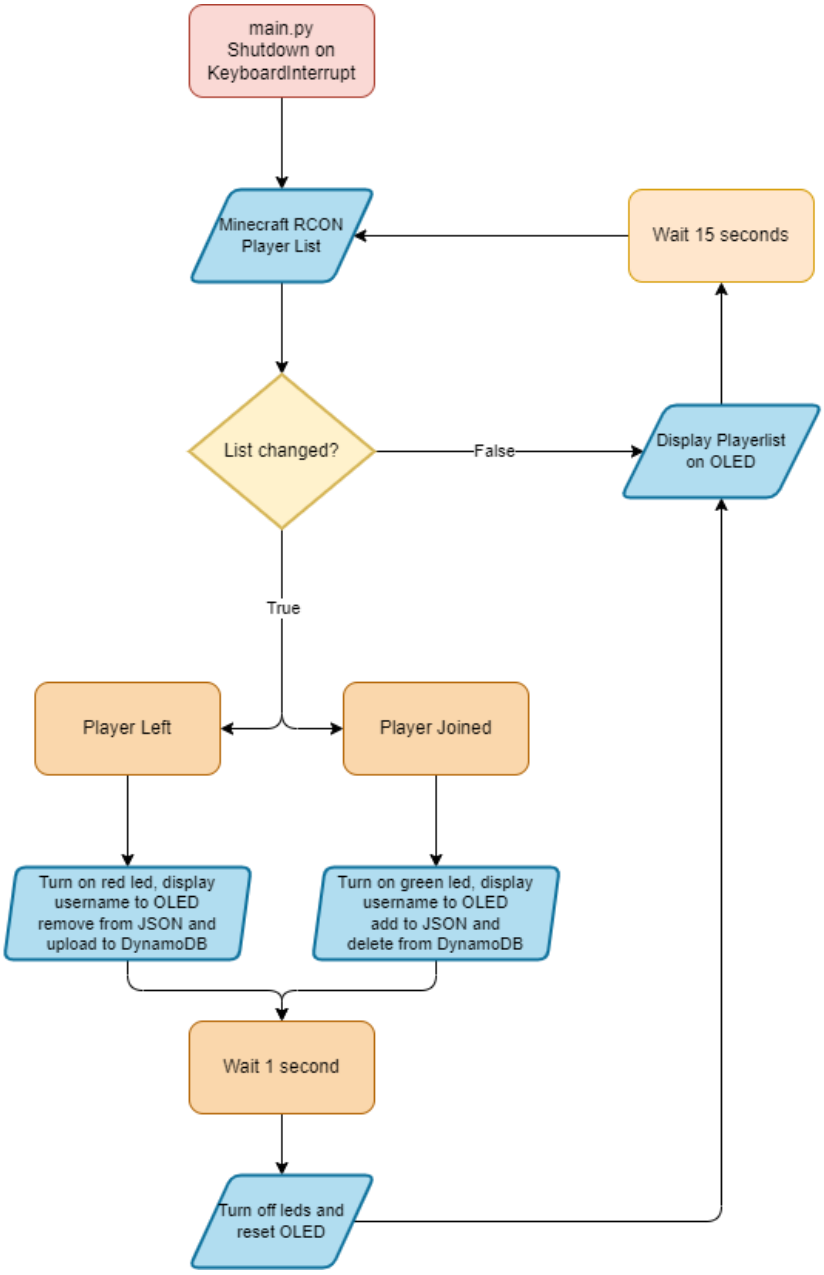

Figure 2 - Minecraft Monitor flowchart

When the main program starts, the RCON client checks the Minecraft Server for the player list and creates a file called 'playerList.json', which stores the current players. Every 15 seconds, it rechecks the player list. When a player leaves or joins, the LED is turned on using the LKM, and the information is displayed on the OLED screen for one second. The JSON file is updated and sent using SCP to the

EC2 instance, and then the player is either removed or added to DynamoDB. Finally, the current player list is displayed on the OLED screen.

Infrastructure

EC2

The EC2 instance was setup as a t2.micro Ubuntu instance, creating the instance generates a pair key to be used to SSH into the instance. AWS provides instructions for this.

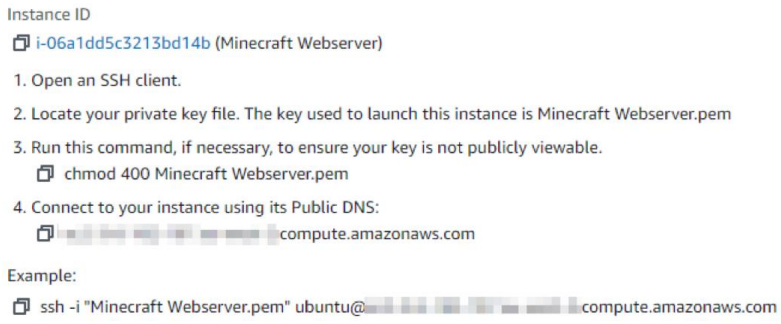

Figure 3 - SSH into EC2 Instance

In the 'Security' tab of AWS, port 80 was opened to inbound traffic to access the webserver. The website was created and controlled by Python using the Flask library. On a GET request for the site, the python script reads the player list JSON file, which is uploaded from the Raspberry Pi. It then gets a JSON file of the offline players by scanning the DynamoDB table using the 'boto3' library.

The player lists are then scanned for new players, and if they are new, their avatars are downloaded locally so that the website loads faster rather than using external links. The players and their avatars are then displayed to the site using the Jinja template engine. Bootstrap 5 was used for the design of the site.

```
{% for chunkUsers in users %}
<div class="row g-0 row-cols-1 row-cols-sm-1 row-cols-md-3 row-cols-lg-3 row-cols-xl-3 row-cols-xxl-3">
    {% if chunkUsers[0] %}
        <div class="col"><img src="{{ url_for('static', filename='images/portraits/%s.png' % chunkUsers[0]) }}" class="rounded img-fluid">
            <h6 class="display-6">{{ chunkUsers[0] }}</h6>
        </div>
    {% endif %}
    {% if chunkUsers[1] %}
        <div class="col"><img src="{{ url_for('static', filename='images/portraits/%s.png' % chunkUsers[1]) }}" class="rounded img-fluid">
            <h6 class="display-6">{{ chunkUsers[1] }}</h6>
        </div>
    {% endif %}
    {% if chunkUsers[2] %}
        <div class="col"><img src="{{ url_for('static', filename='images/portraits/%s.png' % chunkUsers[2]) }}" class="rounded img-fluid">
            <h6 class="display-6">{{ chunkUsers[2] }}</h6>
        </div>
    {% endif %}
</div>
{% endfor %}
```

Figure 4 - Online users being loaded in Jinja template

Figure 5 - Offline users displayed with Jinja template

Again, a '.env' file was used for the credentials and file locations for the same reasons as stated earlier.

To serve the application, Gunicorn WSGI was used with 'systemd' so that it boots on launch of the EC2 instance. Nginx was then used to accept and route requests to Gunicorn.

```
[Unit]
Description=Gunicorn instance for a minecraft web server
After=network.target
[Service]
User=ubuntu
Group=www-data
WorkingDirectory=[redacted]MinecraftWebserver
ExecStart=[redacted]MinecraftWebserver/venv/bin/gunicorn -b localhost:8000 app:app
Restart=always
[Install]
WantedBy=multi-user.target
```

Figure 6 - MinecraftWebserver.service

```
upstream minecraftwebserver {
    server 127.0.0.1:8000;
}
```

Figure 7 - /etc/nginx/sites-available/default

```
location / {
        # First attempt to serve request as file, then
        # as directory, then fall back to displaying a 404.
        proxy_pass http://minecraftwebserver;
}
```

Figure 8 - /etc/nginx/sites-available/default

DynamoDB

The table 'mc_player_status' was created and had two IAM users set up to interact with it programmatically. User 'python-read' was created for the EC2 instance and only had the permission policy 'AmazonDynamoDBReadOnlyAccess', meaning it can only read the database. User 'python-rw' was created for the python script on the Raspberry Pi and only has the permission policy 'AmazonDynamoDBFullAccess', meaning it can read and write to the database.

Limiting the permissions of the IAM users to precisely what they need means less damage can be done to the AWS account if the credentials are leaked.

Conclusion

The goal of this project was met as all functions of Minecraft Monitor work as expected.

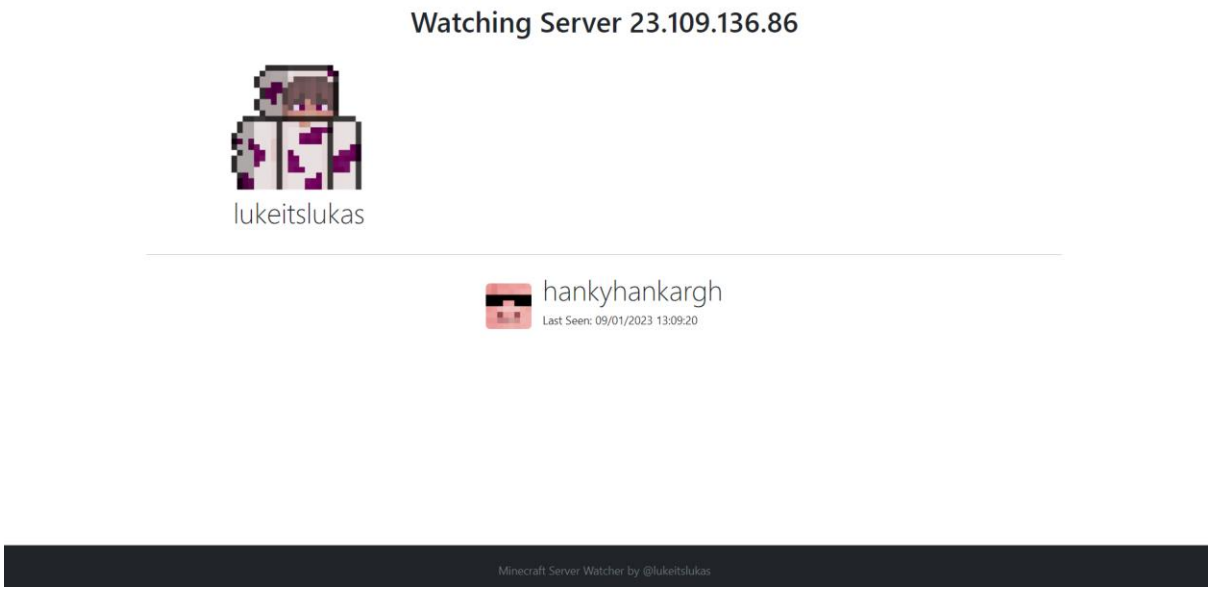

Figure 9 - Minecraft Monitor Webserver with one person logged in

All three sections, hardware, software and infrastructure are working together. Additionally, security has been considered, with credentials being stored in environment files and IAM users being given limited permissions. Finally, the LKM developed has been handled properly, with it being released when the program ends or crashes, allowing another program to use it.

Potential Future Work

The Minecraft Server being monitored uses a service called 'Multicraft'. A possible alternative project using the same hardware and infrastructure could be to use Multicraft to monitor the hardware the server is using, such as the RAM, CPU and Storage.

A further stress test could be conducted on the application if more Minecraft accounts were available. The application's scalability could be estimated by these accounts leaving or joining simultaneously.

References

ada, l., 2012. *Python Usage | Monochrome OLED Breakouts.* [Online]
Available at: https://learn.adafruit.com/monochrome-oled-breakouts/python-usage-2
[Accessed 3 January 2023].

Amazon Web Service, 2023. *DynamoDB examples using SDK for Python (Boto3).* [Online]
Available at: https://docs.aws.amazon.com/code-library/latest/ug/python_3_dynamodb_code_examples.html
[Accessed 3 January 2023].

DataDog, 2023. *IoT Monitoring | DataDog.* [Online]
Available at: https://www.datadoghq.com/solutions/iot-monitoring/
[Accessed 10 January 2023].

Dynatrace, 2023. *Internet of Things (IoT) monitoring.* [Online]
Available at: https://www.dynatrace.com/solutions/internet-of-things/
[Accessed 5 January 2023].

Flask, 2010. *Quickstart - Flask Documentation.* [Online]
Available at: https://flask.palletsprojects.com/en/2.2.x/quickstart/#rendering-templates
[Accessed 3 January 2023].

Huiyeon, K., 2020. *Step-by-step visual guide on deploying a Flask application on AWS EC2.* [Online]
Available at: https://medium.com/techfront/step-by-step-visual-guide-on-deploying-a-flask-application-on-aws-ec2-8e3e8b82c4f7
[Accessed 3 January 2023].

Jinja, 2007. *Jinja Documentation.* [Online]
Available at: https://jinja.palletsprojects.com/en/3.1.x/
[Accessed 3 January 2023].

Jones, K., 2001. Loadable Kernel Modules. *login: The Magazine of USENIX and SAGE,* 26(7), pp. 42-49.

Splunk, 2022. *What Is IoT Monitoring?.* [Online]
Available at: www.splunk.com/en_us/data-insider/iot-monitoring.html
[Accessed 10 January 2023].