Cybersecurity Pedagogy & Practice Journal

Volume 2, No. 2
September 2023
ISSN: 2832-1006

In this issue:

Information Systems & Computing Academic Professionals

The **Cybersecurity Pedagogy and Practice Journal** (**CPPJ**) is a double-blind peer-reviewed academic journal published by **ISCAP** (Information Systems and Computing Academic Professionals). Publishing frequency is two times per year. The first year of publication was 2022.

CPPJ is published online (https://cppj.info). Our sister publication, the proceedings of the ISCAP Conference (https://proc.iscap.info) features all papers, panels, workshops, and presentations from the conference.

The journal acceptance review process involves a minimum of three double-blind peer reviews, where both the reviewer is not aware of the identities of the authors and the authors are not aware of the identities of the reviewers. The initial reviews happen before the ISCAP conference. At that point, papers are divided into award papers (top 15%), and other accepted proceedings papers. The other accepted proceedings papers are subjected to a second round of blind peer review to establish whether they will be accepted to the journal or not. Those papers that are deemed of sufficient quality are accepted for publication in the CPPJ journal.

While the primary path to journal publication is through the ISCAP conference, CPPJ does accept direct submissions at https://iscap.us/papers. Direct submissions are subjected to a double-blind peer review process, where reviewers do not know the names and affiliations of paper authors, and paper authors do not know the names and affiliations of reviewers. All submissions (articles, teaching tips, and teaching cases & notes) to the journal will be refereed by a rigorous evaluation process involving at least three blind reviews by qualified academic, industrial, or governmental computing professionals. Submissions will be judged not only on the suitability of the content but also on the readability and clarity of the prose.

Currently, the acceptance rate for the journal is under 35%.

Questions should be addressed to the editor at editorcppj@iscap.us or the publisher at publisher@iscap.us. Special thanks to members of ISCAP who perform the editorial and review processes for CPPJ.

2023 ISCAP Board of Directors

Cybersecurity Pedagogy and Practice Journal

IoT Security Vulnerabilities Analysis by Reverse Engineering: A Face-recognition IoT Application-based Lab Exercises

Sam Elfrink
selfrink3s@semo.edu

Mario Alberto Garcia
mgarcia@semo.edu

Xuesong Zhang
xzhang@semo.edu

Zhouzhou Li
zli2@semo.edu

Department of Computer Science
Southeast Missouri State University
Cape Girardeau, MO, US

Qiuyu Han
2003138@hlju.edu.cn
Heilongjiang University
Harbin, Heilongjiang, CN

Abstract

The rapid growth of the Internet users and the proliferation of IoT devices in recent years has created a significant need for vulnerability detection and mitigation in these devices and their applications. Exposing computer science and cybersecurity students to these skills can help them strengthen their competencies in the industry. One approach that can be used to achieve this objective is reverse engineering, which involves gaining a thorough understanding of the relationship between the individual components of an IoT application. This paper presents lab exercises that teach students the concepts and practical techniques of reverse engineering for the purpose of detecting and mitigating vulnerabilities in IoT devices. The lab exercises are based on a real facial recognition web application hosted on a small IoT device, and they use both manual exploration and automated tools to provide students with a systematic and comprehensive understanding of reverse engineering. These well-designed, hands-on labs can meet the practical needs of cybersecurity education and inspire heuristic learning on difficult cybersecurity topics such as reverse engineering.

Keywords: Cybersecurity, Reverse Engineering, Internet of Things, Artificial Intelligence, Face Recognition, Re-engineering.

1. INTRODUCTION

In recent years, the number of users with access to the Internet has increased dramatically. An estimated one million new users join the Internet every day. And six billion people are projected to be connected to the Internet by the end of 2022, a 1 billion increase from the 5 billion people in 2020 (Morgan, 2020). Internet of Things (IoT) are also becoming prevalent. In 2015, regarding the number of connected devices by 2020, the low estimate was 18 billion, the most bullish forecast stated 50 billion devices (Lueth, 2015). Reputable companies such as Cisco, Intel, IDC, Gartner, etc., provide similar and higher estimates recently.

This substantial growth of Internet users and Internet capable devices has increased the risk of cyberattacks. Cybercrime costs are estimated over six trillion USD globally in 2021, and that number is expected to grow 15 percent each year, reaching 10.5 trillion USD in 2025 (Morgan, 2020). Because of this increase in both users and attacks, the need for security vulnerability analysis has never been more vital, especially for web applications. According to a global threat analysis report produced by Radware, the average number of blocked malicious web application requests increased by 88% from 2020 to 2021, with 75 percent of those attacks performed via broken access control and injection (Radware, 2022).

The massive increase in Internet access has also led to large increases in the quantity and diversity of Internet capable devices. The term "Internet of Things" (IoT) has been used to describe this phenomenon. Shwartz, Mathov, Bohadana, Elovici, & Oren (2018) define IoT as "a network of smart electronic devices with Internet connectivity." Nowadays a home containing a variation of IoT devices (also known as smart devices) that provide a variety of functions to the consumer is common. The introduction of these devices also introduces new security concerns and vulnerabilities.

As cybercrime and security vulnerabilities increase, Cybersecurity and Computer Science professionals need the skills to mitigate threats to software applications. They must be proactive in detecting and patching vulnerabilities before malicious actors can exploit the system.

As students graduate and enter the industry, vulnerability detection and mitigation skills will be invaluable to corporations, as patching vulnerabilities before they are exploited can save corporations millions of dollars as well as their reputation as a reliable company. While some software engineers can be tasked with creating new systems, many are tasked with maintaining, testing, reusing, and integrating existing systems (Canfora & Penta, 2007). New hires probably will be asked to work on systems that are already established, thus the ability to adapt and understand code that is not their own and test it for vulnerabilities is an important skill to possess. In order to accomplish this, they must be able to identify and comprehend the different components in a system and understand the relationship between them. However, most curricula only focus on designing software applications from scratch without regarding the common situation in the industry.

There are a variety of tools and methods that can be used for software vulnerability detection, such as automated dynamic scanning (tool-based), static analysis (tool-based), and Reverse Engineering. While tools can be useful, their ease of use and quick results can lead to a lack of a holistic understanding of how a piece of software operates. Thus, courses that only utilize tools can run the risk of focusing on teaching how to use the tools instead of how to understand the underlying security and software principles.

Reverse Engineering can be defined as "the process of analyzing a subject system to identify the system's components and their interrelationships and create representations of the system in another form or at a higher level of abstraction" (Chikofsky & Cross, 1990). It has a vast number of goals, such as managing system complexity, creating alternate views, recovering lost understanding of a system, and existing software maintenance. It can also be used to audit the security and vulnerabilities of an application (this is a unique step in Secure Software Development, so it is not equivalent to software maintenance).

From an offensive perspective, in many cases attackers or pen-tester have limited or incomplete access to the application's entire framework. For instance, an attacker or pen-tester may be able to view the front-end source code of an application but will not easily be aware of the backend code. However, through the process of Reverse Engineering, the attacker or pen-tester can gain an understanding of how the entire application operates.

There are three basic approaches on how to Reverse Engineer a system. The first is white box analysis, which is a static approach that does not require running the application (Ali, 2005). This approach is often used when all the systems components are easily available. The second approach is black box analysis, which uses inputs to check the behavior of the application/program. This approach is used when the user has less access to the backend of the system and must use inputs to determine the hidden components of the system. Finally, gray-box analysis utilizes both white box and black box in unison to evaluate the system, using each approach on various parts of the system.

Real-world software systems are in constant need of modification and improvement due to new user requirements, modified business models, and changing legislation (Canfora, Penta, & Cerulo, 2011). When applied to IoT-based web application security, Reverse Engineering can be used to develop a detailed understanding of how the application is constructed and how it operates. An IoT device hosting a web application can be broken down into 5 simplified layers: the physical, link, network, transport, and web app layers (Fig. 1). To have a complete understanding of a system's vulnerabilities, each layer of the application must be considered. Each layer has different attack surfaces and capabilities, and together their specific vulnerabilities can be combined to form a complete picture of the application's potential security concerns.

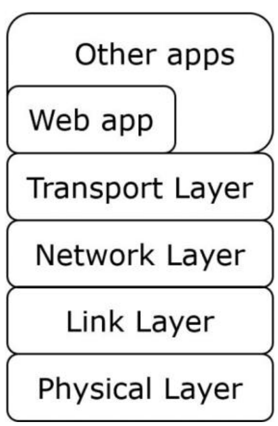


Fig. 1. Layer Diagram of Web Application

There are several challenges to effectively creating a teaching curriculum for Reverse Engineering an IoT-based web application. First, the application and device must be at an appropriate level of complexity to match the abilities and knowledge of the students. Because Reverse Engineering often requires making inferences about the structure of an application and complicated analysis of the system, many applications are too complex to suit a curriculum for beginners. In addition, the IoT device used in the curriculum must be affordable and easily accessible so that faculty members and students can use them effectively. Lastly, the testing environment must be easily assembled so that entry into the exercises is not too difficult.

The purpose of this paper is to provide a curriculum that students can utilize to gain hands-on experience with Reverse Engineering on a real-world IoT-based web application that integrates a face-recognition Artificial Intelligence (AI) model. This is accomplished by providing general background information on the topics discussed to promote a better understanding, followed by 5 hands-on labs that identify a variety of vulnerabilities of a facial recognition web application hosted by the ESP32-CAM IoT device. The vulnerabilities encompass multiple layers of the application and are discovered and mitigated through the process of Reverse Engineering and re-engineering. Thus, giving students the opportunity to practice these vital industry skills on a real application.

The remainder of this paper is organized as follows. In the 'Literature Review' section, we review the current courseware or labs designed for teaching Reverse Engineering. In the 'Background' section, the Internet of Things (IoT) device-based face recognition Web platform will be introduced, and its advantages will be explained. In the 'Vulnerabilities Found by Reverse Engineering' section, the specific vulnerabilities identified in the course will be discussed followed by an outline of the step-by-step processes taken to assess each vulnerability that was identified. A summary of what areas can be improved and expanded is provided in the 'Future Work' section, and the 'Conclusion' section provides a general summary of the paper's contents and findings.

2. LITERATURE REVIEW

The concept of Reverse Engineering system began to get traction in the beginning of the 1990s and a variety of methods and approaches have been proposed as systems have shifted to web-based user interfaces (Müller, Jahnke, Smith, Storey, Tilley, & Wong, ,2000). Early papers provide a helpful summary of the history of Reverse Engineering and the different methods and approaches involved. However, they only provide conceptual information and do not provide any sort of tutorial to learn Reverse Engineering.

Shwartz et al. (2018) provides a detailed methodology and tutorial for Reverse Engineering and security vulnerability evaluation of what are considered “full stack OS devices” that contain a modern operating system such as Linux. These devices divide execution into kernel mode and user mode. However, this method does not address partial stack OS devices that have specially designed real-time operating systems, or devices that execute compiled instructions directly with no operating system. It provides a narrow focus of Reverse Engineering from a physical access perspective, dealing with device’s firmware memory images rather than the applications that are hosted by the devices as well.

Ali (2015) gives a detailed argument for the importance of Reverse Engineering for undergraduate software engineering students. While Reverse Engineering can be difficult and time consuming, the process itself is very informative and gives students the ability to understand a system more rapidly and effectively. In addition, it points out that traditionally students are often given the task of designing and implementing new systems in the classroom. While this is a valuable skill, it is also crucial that they be able to understand code written by other programmers and improve upon existing software. Often in industry, companies already have legacy software and are interested in improving it rather than creating an entirely new product. Thus, Reverse Engineering skills are tremendously important. However, the paper merely emphasizes the importance of Reverse Engineering, and does not provide tutorials for students to use to achieve a better understanding of these concepts.

Bellettini et al. outlines the usage of an automatic tool for creating UML (Unified Modeling Language) models for web applications called *WebUml* (Bellettini, Marchetto, & Trentini, 2004.) It describes how the tool can be used for Reverse Engineering by utilizing the rich information that the extracted UML models provide. It provides the source code for a simple XML node construction application that can be used to test the effectiveness of *WebUml*. Thus, while this provides a simple use case for learning to use the tool for Reverse Engineering, the scope of the paper is limited to producing UML diagrams.

Taylor & Collberg (2016) also proposed a tool for teaching Reverse Engineering. It notes that the current instruction materials regarding Reverse Engineering code have a lack of easy tools for students to use. It also notes that students take a substantial amount of time to configure and set up an environment that has the tools necessary to practice Reverse Engineering. The solution provided is an automated code obfuscator tool called Tigress that is combined with a web application. The application allows the instructor to generate custom target programs, which can then be obfuscated with a set level of complexity. The application then creates virtual machines that have the necessary Reverse Engineering tools selected by the instructor. The paper provides two example C programs that can be used, one that checks the current time and prints the variable, and an additional one that adds a password check to the first. A group of students were given this exercise and then polled at the end. While most students reported that the difficulty of solving the challenge was hard, a good percentage also indicated that it was easy, indicating that the student’s previous experience could be a large factor. In addition, most students were able to finish the assignment in a reasonable time, indicating that the proposed application was effective in creating a comfortable environment to practice Reverse Engineering skills.

This paper provides an in-depth framework for students to follow to get hands on experience of Reverse Engineering through the purposes of vulnerability detection and mitigation that to our best knowledge has not been provided by existing works. The application that is examined is hosted by a IoT device with a compact operating system, and the vulnerabilities addressed are not just at the physical layers but encompass multiple layers and incorporate analysis of the applications code and the devices firmware. While automated tools can be helpful aids in Reverse Engineering, relying on them too much keeps students from understanding the process in depth. This paper provides a mixture of automated tools and manual exploration to provide students with a more systematic approach to Reverse Engineering.

3. BACKGROUND

As technology has advanced in recent years, the availability and usefulness of IoT devices has increased dramatically. While smart devices such as Amazon Alexa or smart appliances are popular, IoT devices can also be used for facial recognition purposes as well. Facial recognition is of particular interest because it can be used for a variety of purposes. For instance, IoT devices with facial recognition capabilities can be used for security purposes where authorized individual’s faces are scanned into the system and all other faces considered hostile. In addition, it could be used for recreational home use, such as identify what individuals are in a home to adjust the experience accordingly.

The ESP32-CAM (Wikipedia contributors, 2022) IoT device (Fig. 2) integrates a member of the most popular IoT SoCs, the ESP series (Li, Ren, Chou, Liu, & McAllister, 2022), which offers a good introduction to the concept of IoT device-based face recognition by integrating an AI Model with the IoT device. The ESP32-CAM interfaces with the computer via micro-USB connection. It can be purchased for $5 to $10 per device depending on the vendor. The inexpensive cost of the device and the ease of deployment makes it a suitable option for introductory exploration into the concepts of IoT device-based face recognition.

Fig. 2. ESP32-CAM IoT Device

The device contains a dual-core 32-bit ESP32-S CPU, 520 KB of SRAM, and 4M of PSRAM (Fig. 2). Furthermore, it has an 802.11b/g/n Wi-F0 BT/BLE System on Chip (SoC) module and a camera that supports OV2640 and OV7670, giving it both Wi-Fi and image capability.

There are a variety of pre-prepared programs that can be easily implemented on the device, including a face recognition web platform (Li, Chou, & McAllister, 2022). The web application provides the user with the ability to register faces to the application and then recognize registered users when they come into the view of the camera. The device sends a continuous stream of video footage back to the application, acting as a live feed security camera as well.

The application also allows the user to modify a variety of settings for the camera display, such as saturation, brightness, and the frame size of the stream. The program is also designed to log the facial details of recognized individuals. The logs are transported via USB connection and are displayed in the serial monitor, recording the ID of individuals that have come into the view of the device's camera.

The benefits of the web platform for facial recognition are apparent, as the user can easily access the application on any Internet capable device and can access it from any location if they have access to the network and knowledge of the web application's IP address. In addition to being deployed on an existing local network, the device's code can be easily modified to create a unique Wi-Fi AP that has its own SSID and password credentials.

However, the ability for the device to connect the facial recognition and streaming capabilities to a web application presents some unique challenges for security as well. This paper aims to address these security vulnerabilities and demonstrate how students can learn essential security principles by evaluating and mitigating the vulnerabilities that are exposed. Although it may appear simple because of the ease of deployment, the facial recognition web application offers significant depth. Fig. 3 is a model created by Microsoft's STRIDE application that shows the data flow and trust boundaries during the device's operation.

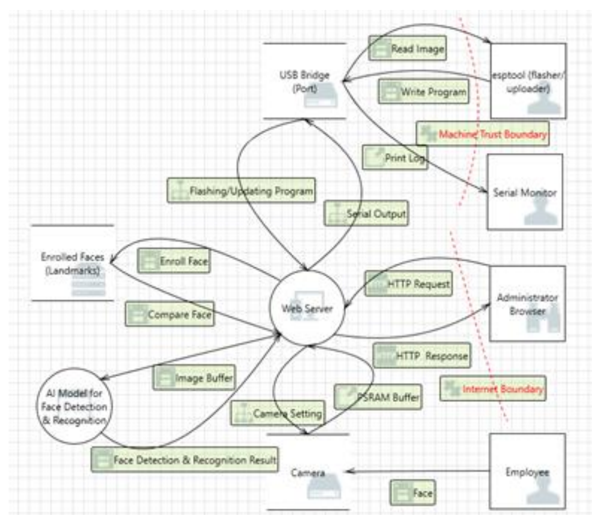


Fig. 3. STRIDE (Microsoft) Data Flow Model for the Face-Recognition Web Application

In addition to the devices, there are a variety of common items and tools that will need to be assembled to reproduce the methods and concepts introduced in this paper. Fig. 4 outlines required hardware and tools in detail. ESP32-CAM is the target IoT device hosting a face-recognition Web App. Its camera can take photo of the user and send the user's face image to an AI model for analysis. Then the AI model can conduct face detection and face recognition (if the user registered before). The USB cable is used to connect the IoT device to the computer, where the Arduino IDE (Fezari & Al, 2018) is installed to prepare, compile, and upload the face-recognition application to the device. Also, the computer is the output peripheral of the device, where the device log can be printed out. cscope is the tool used to build the code database including the App's source code (4 files) and thousands library files. cscope must work with a text editor to fully function. Finally, the Micro STRIDE tool will be used to do threat modeling (Fig. 3).

Hardware:

A ESP32-CAM IoT device (including a mini camera)

A USB A to micro-USB B cable

Tools:

Arduino IDE with the ESP32 add-on

cscope

Text editor such as vi

Microsoft STRIDE (Threat Modeling)

Fig. 4. Curriculum Hardware and Software Requirements

4. VULNERABILITIES FOUND BY REVERSE ENGINEERING

Through Reverse Engineering, five categories of vulnerabilities were found hidden in ESP32-CAM IoT devices when they are programmed with the example face recognition application. To set up a proper lab environment to host the web application, follow the instructions detailed in the attached "Lab Environment Setup" document. The following sections provide detailed steps for the vulnerabilities that were detected and mitigated with Reverse Engineering and re-engineering.

Figure out the hard-coded credential

This lab demonstrates how physical access to the UART/USB port on the IoT device can be exploited by an attacker to gain access to the hardcoded credentials. Students will need to Reverse Engineer the application upload process (from a development host to the IoT device) to figure out the tools for them to fetch the binary image from the IoT device (in reverse direction). Then, the students will need to Reverse Engineer the binary image to figure out where the hardcoded credential was saved. The Arduino IDE is simply a wrapper that provides a graphical user interface for ease of use, but at the core it calls a code compiler and uploader to upload the binary executable to the device. The attacker can upload code to a similar device with a verbose log output to understand what tools the program is using. Through Reverse Engineering, the program tools used for flashing to the device can be discovered. Then, further exploration can be done to determine the parameters of the tool to discover how it can read the flash as well. Then with the correct parameters for the flash reader, contents of the flash can be copied to a file and examined for hard-coded credentials and sensitive information.

The SSID and password are hard coded into the application's code. If physical access to the device is gained, these sensitive credentials could be compromised through analysis of the binary code. We know that the binary code was cross compiled in the Arduino IDE and flashed to the device via USB port. The same uploading tool could be used to extract the binary code from the device, thus reversing the direction and exploiting the hard-coded credential vulnerability. Here are the steps:

1. In the Arduino IDE, navigate to File -> Preferences, and make sure that verbose output is checked for compilation and uploading (Fig. 5).

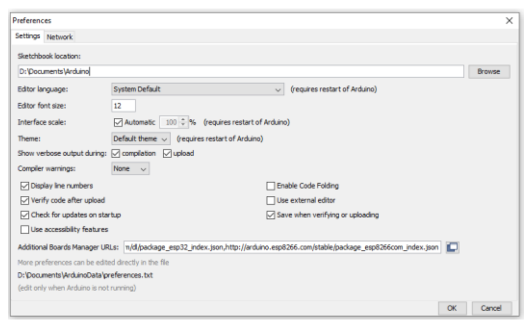


Fig. 5. Arduino IDE Preference Settings

2. Compile and upload the program, look at the verbose log to see what underlying tools are called to upload the compiled program to the IoT device and where they are installed. From the output, we can see that "D:\Documents\ArduinoData\packages\esp32\tools\esptool_py\2.6.1/esptool.exe" is the uploading tool (Fig. 6).

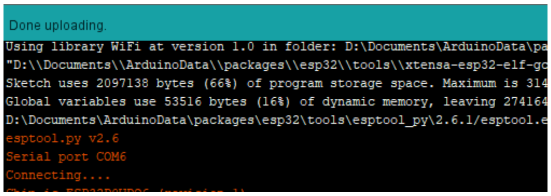


Fig. 6. Verbose Output for Uploading

3. Locate "esptool.exe" and the underlying esptool.py (Fig. 7).

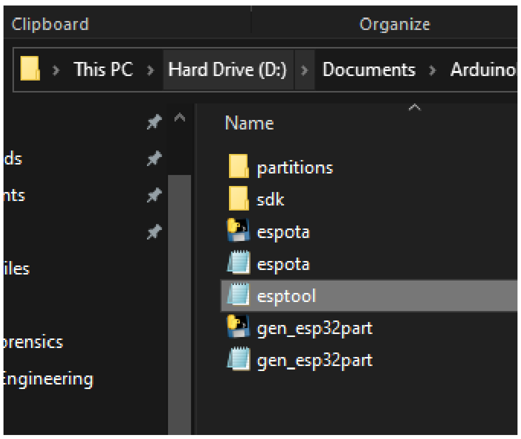


Fig. 7. Esptool.py File Location

4. Run the esptool.py to see the options and get more information. The write_flash option was used to upload the compiled program to the IoT device. To download the active image from the device (in a reverse direction), note the "read_flash" parameter (Fig. 8).

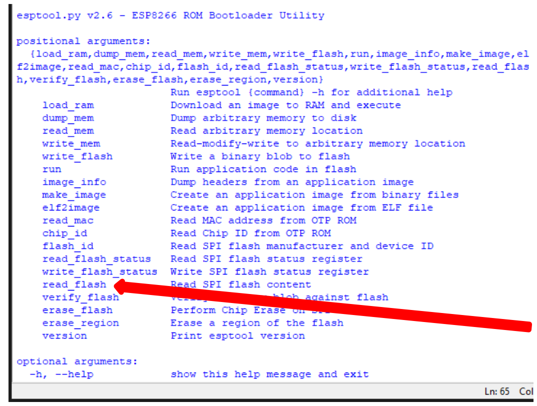


Fig. 8. Esptool.py Parameters

5. Run the "esptool.py" (Fig. 9) again with the "read_flash" parameter to see the sub-options. 'address', 'size', and 'filename' are mandatory sub-options.

6. Enter in the parameters "-b 921600 read_flash 0 0x400000 targetImg". The '-b' option will set the USB communication's baud rate to 921600, which is the maximum speed. '0' is the starting address of the image download. '0x400000' is the image size – ESP32-CAM has a 4MB flash. And the "targetImg" option specifies the image output filename (Fig. 10).

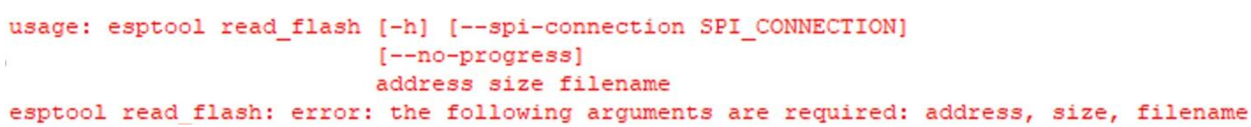


Fig. 9. Esptool.py read_flash Parameters

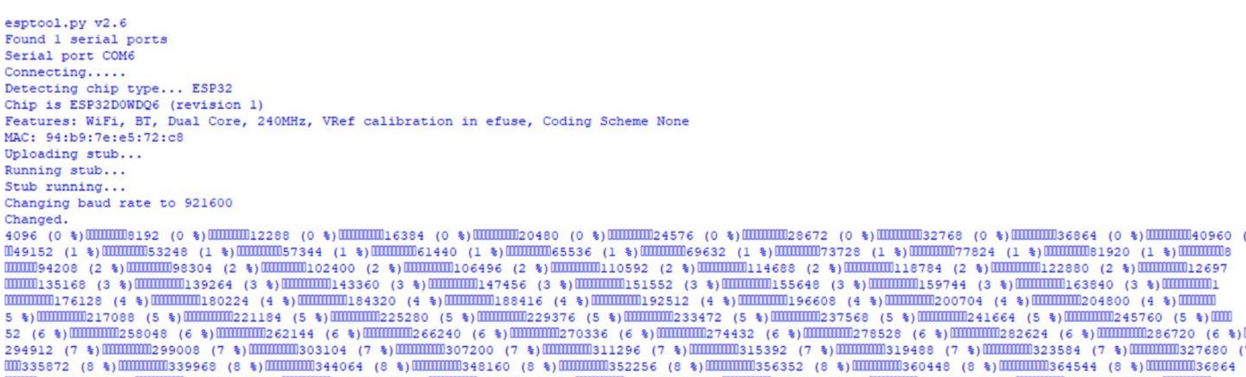


Fig. 10. Esptool.py Running Results

7. Once it has finished running, we can now navigate back to the folder containing "esptool.py" and should see the "targetImg" file that was generated (Fig. 11).

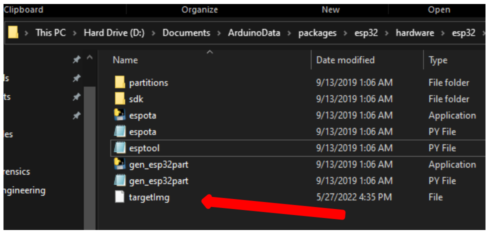


Fig. 11. Downloaded ESP32-CAM Image

8. Take the targetImg file and upload it to a Kali Linux VM. Using a terminal, navigate to the folder that contains the file and type the command "strings targetImg" to derive all readable strings from the binary image (Fig. 12).

Fig. 12. Commands to Derive Strings from a Binary Image

9. So many strings are derived from the "targetImg" file (Fig. 13).

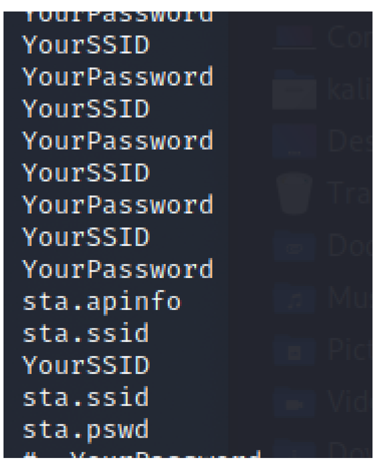


Fig. 13. Embedded Strings in targetImg

10. The SSID can be scanned by a cell phone. The idea is if we can find the SSID string embedded in targetImg, we probably will be able to find its password around it. The grep command along with the context searching options (2 lines 'B'efore and 2 lines 'A'fter the keyword 'SSID') can be used to search for strings around the SSID string (Fig. 14). This will

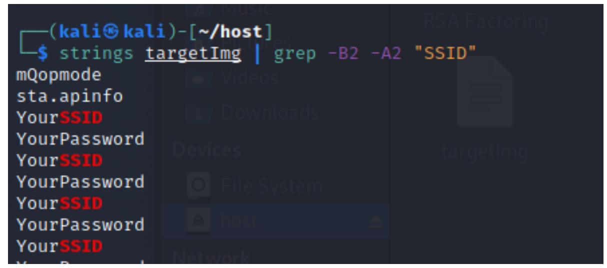


Fig. 14. Narrow down the Password Searching by "grep"

11. Thus, by Reverse Engineering the compiler and uploader that are called by the Arduino IDE, secrets hardcoded on the device can be extracted.

Trigger the application corruption by a buffer overflow

Front-end code of a Web application is usually the start point for attackers to perform Reverse Engineering against the application. Testing a variety of inputs can help discover vulnerabilities. Through the process of black box Reverse Engineering, a buffer overflow vulnerability in the framesize input variable can be detected. White box Reverse Engineering can then be used to inspect the code and find the area that pertains to the handling of the control variables of the application.

In addition, discovering this vulnerability demonstrates the importance of manual exploration, as an automated scan for vulnerabilities would not have necessarily discovered this vulnerability.

Here are the steps taken to discover the vulnerability:

1. Manually explore the web application. Note that the applications control panel offers a variety of user preferences that can be adjusted such as resolution, quality, brightness, etc. (Fig. 15).

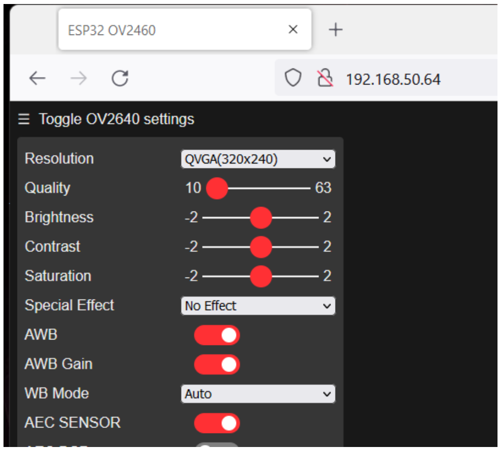


Fig. 15. Web Application Control Panel

2. Click the "Resolution" drop-down menu and select the largest setting. Then click "Start Stream". You should see that the stream window size is now much larger than the default (Fig. 16).

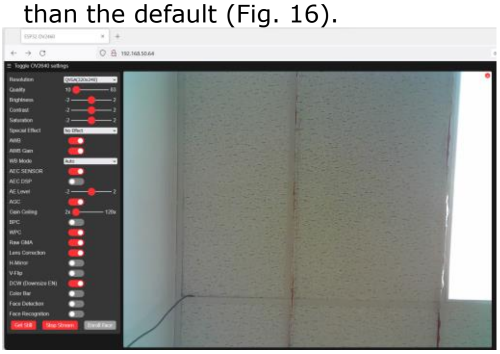


Fig. 16. Web Application Large Frame size

3. Now that we know that users can send control requests to the backend code that modifies aspects of the application, an analysis of the backend code can be performed to look for vulnerabilities.

4. By a quick look through the code, we can see that the "app_httpd.cpp" file contains all the handlers for the application: "stream_handler()," "cmd_handler()," "status_handler(),", "capture_handler()", and "index_handler()."

5. The handlers are initialized by the function "startCameraServer()". Fig. 17 shows the code fragment of the startCameraServer() function.

```
void startCameraServer(){
    httpd_config_t config = HTTPD_DEFAULT_CONFIG();

    httpd_uri_t index_uri = {
        .uri       = "/",
        .method    = HTTP_GET,
        .handler   = index_handler,
        .user_ctx  = NULL
    };

    httpd_uri_t status_uri = {
        .uri       = "/status",
        .method    = HTTP_GET,
        .handler   = status_handler,
        .user_ctx  = NULL
    };

    httpd_uri_t cmd_uri = {
        .uri       = "/control",
        .method    = HTTP_GET,
        .handler   = cmd_handler,
        .user_ctx  = NULL
    };

    httpd_uri_t capture_uri = {
        .uri       = "/capture",
        .method    = HTTP_GET,
        .handler   = capture_handler,
        .user_ctx  = NULL
    };

   httpd_uri_t stream_uri = {
        .uri       = "/stream",
        .method    = HTTP_GET,
        .handler   = stream_handler,
        .user_ctx  = NULL
    };
```

Fig. 17. Code fragment of "startCameraSever()" Function

6. From the names of the handlers, the "cmd_handler()" function should control the user commands on the control panel. Inspecting the code inside of the handler confirms this as it contains all the different control panel variables (Fig. 18).

```
}
else if(!strcmp(variable, "quality")) res = s->set_quality(s, val);
else if(!strcmp(variable, "contrast")) res = s->set_contrast(s, val);
else if(!strcmp(variable, "brightness")) res = s->set_brightness(s, val);
else if(!strcmp(variable, "saturation")) res = s->set_saturation(s, val);
else if(!strcmp(variable, "gainceiling")) res = s->set_gainceiling(s, (gainceiling_t)val);
else if(!strcmp(variable, "colorbar")) res = s->set_colorbar(s, val);
else if(!strcmp(variable, "awb")) res = s->set_whitebal(s, val);
else if(!strcmp(variable, "agc")) res = s->set_gain_ctrl(s, val);
else if(!strcmp(variable, "aec")) res = s->set_exposure_ctrl(s, val);
else if(!strcmp(variable, "hmirror")) res = s->set_hmirror(s, val);
else if(!strcmp(variable, "vflip")) res = s->set_vflip(s, val);
else if(!strcmp(variable, "awb_gain")) res = s->set_awb_gain(s, val);
else if(!strcmp(variable, "agc_gain")) res = s->set_agc_gain(s, val);
else if(!strcmp(variable, "aec_value")) res = s->set_aec_value(s, val);
else if(!strcmp(variable, "aec2")) res = s->set_aec2(s, val);
else if(!strcmp(variable, "dcw")) res = s->set_dcw(s, val);
else if(!strcmp(variable, "bpc")) res = s->set_bpc(s, val);
else if(!strcmp(variable, "wpc")) res = s->set_wpc(s, val);
else if(!strcmp(variable, "raw_gma")) res = s->set_raw_gma(s, val);
else if(!strcmp(variable, "lenc")) res = s->set_lenc(s, val);
else if(!strcmp(variable, "special_effect")) res = s->set_special_effect(s, val);
else if(!strcmp(variable, "wb_mode")) res = s->set_wb_mode(s, val);
else if(!strcmp(variable, "ae_level")) res = s->set_ae_level(s, val);
else if(!strcmp(variable, "face_detect")) {
    detection_enabled = val;
    if(!detection_enabled) {
        recognition_enabled = 0;
    }
```

Fig. 18. Code Fragment of "cmd_handler()" Function

7. By looking at the code, we can see there is no input validation for the "framesize" variable in the cmd_handler() function. It simply uses the value of the "val" parameter that is passed from the user request. So, what if the user tries setting the framesize to a negative number?

```
int val = atoi(value);
sensor_t * s = esp_camera_sensor_get();
int res = 0;

if(!strcmp(variable, "framesize")) {
```

```
        if(s->pixformat == PIXFORMAT_JPEG) {
            res = s->set_framesize(s, (framesize_t)val);
        }
    }
```

8. From this block of code, the "/control" URL via a HTTP_GET request can be used to trigger the "cmd_handler()" function (Fig. 19).

```
httpd_uri_t cmd_uri = {
        .uri       = "/control",
        .method    = HTTP_GET,
        .handler   = cmd_handler,
        .user_ctx  = NULL
    };
```

Fig. 19. Code Fragment of "startCameraServer()" Function

9. With this information, we can conclude that we can bypass the drop-down menu of the application and enter in the request directly into the webpage's search box by combining the designation for the command handler, the control variable, and the control value. Thus, the attack vector http://{IP}/control?var=framesize&val=x where the "IP" is replaced with the IP address of the App and the "x" can be replaced with any value. This will allow for values outside of the drop-down box parameters to be input into the system.

10. Go to the application and click "start stream" and verify it is working properly. Note the size of the stream window (Fig. 20).

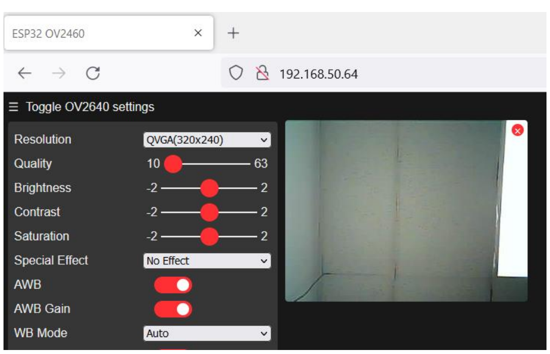


Fig. 20. Web Application Stream

11. Enter the attack vector with "x" being set to the value 10. A blank page will load, click back to the application. You should see that the window is bigger, indicating that request was successful in modifying the application's frame size (Fig. 21). Now a variety of strange inputs can be tried to test the application for bugs.

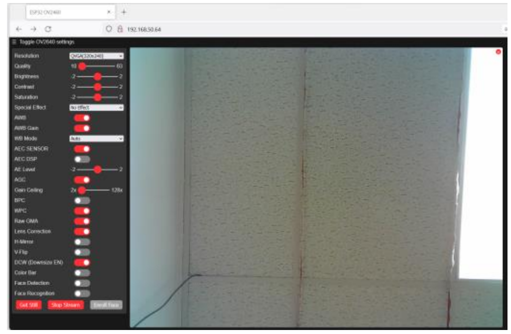


Fig. 21. Web Stream Enlarged

12. Try a negative value for the parameter. The application fails to display a window at all, revealing a successful attack to the system (Fig. 22).

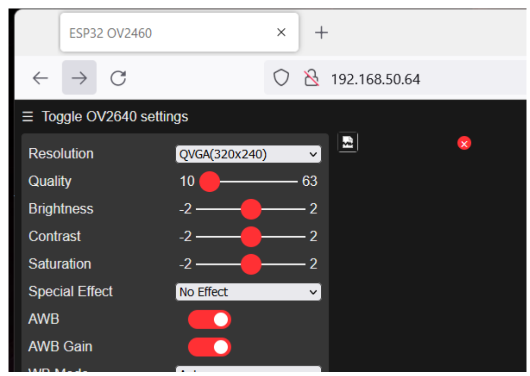


Fig. 22. Web Stream Broken

13. Try a ridiculous big value for the parameter. You should see that this request causes the stream to be unresponsive and timeout. A look at the serial monitor logs reveals that it has caused an exception and the system is continually rebooting (Fig. 23).

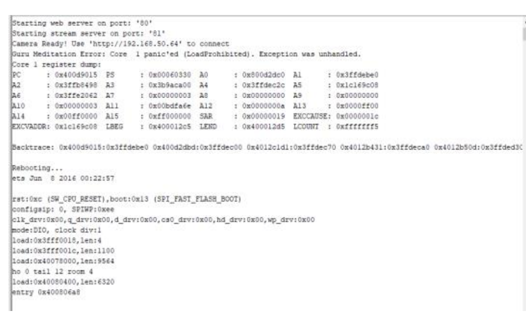


Fig. 23. Serial Monitor Error Log

14. Try a smaller value that is still larger than 10. You should see that the stream is broken and full of unintelligible bars (Fig. 24). In this case since 12 appears to be just slightly over the given threshold, the overflow resulted in just the stream data being corrupted and did not crash the entire application.

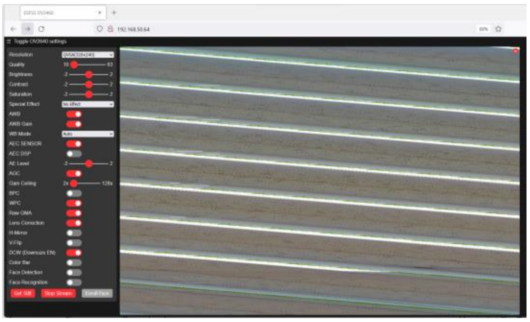


Fig. 24. Frame size Buffer Overflow

15. Try to enter in a decimal value next, such as 5.5. The display should simply round the number and display the appropriate rounded value (Fig. 25).

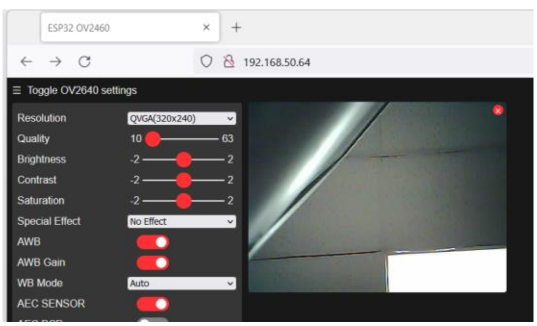


Fig. 25. Decimal Value for Frame size

16. Thus, the proper parameters for the framesize variable can be inferred: do not include negative numbers or numbers greater than 12. However, we do not yet know how large the numbers can be.

```
typedef struct {
    framesize_t framesize;//0 - 10
    uint8_t quality;//0 - 63
    int8_t brightness;//-2 - 2
    int8_t contrast;//-2 - 2
    int8_t saturation;//-2 - 2
    int8_t sharpness;//-2 - 2
    uint8_t denoise;
    uint8_t special_effect;//0 - 6
    uint8_t wb_mode;//0 - 4
    uint8_t awb;
    uint8_t awb_gain;
    uint8_t aec;
    uint8_t aec2;
    int8_t ae_level;//-2 - 2
    uint16_t aec_value;//0 - 1200
    uint8_t agc;
    uint8_t agc_gain;//0 - 30
    uint8_t gainceiling;//0 - 6
    uint8_t bpc;
    uint8_t wpc;
    uint8_t raw_gma;
    uint8_t lenc;
    uint8_t hmirror;
    uint8_t vflip;
    uint8_t dcw;
    uint8_t colorbar;
} camera_status_t;
```

Fig. 26. cscope Findings

17. We can use the 'cscope' tool to collect library code installed for Arduino IDE and ESP32 add-on. Along with the 4 source files, we can build a code database, where we can use vi to search for all the occurrences of an interesting variable through the code database. Then, we can get more information about that variable. For example, by searching for the keyword 'framesize', we can find a struct, where the valid values per that variable are shown in the comments (Fig. 26).

18. Thus, the valid inputs for the framesize parameter are between 0-10. Negative numbers cause a display failure and values greater than 10 cause the stream data to be corrupted due to a buffer overflow.

5. FUTURE WORK

While this paper identifies a variety of vulnerabilities in the face-recognition Web App for the ESP32-CAM IoT device, further exploration of the code and additional testing could reveal additional vulnerabilities that were not identified in this paper. E.g., we will cover "vulnerability detected by auto scan: X-frame options header not set", "AI model vulnerability", and "3rd party library vulnerability" lab exercises in another paper. In addition, the "Data Corruption" and "AI Model Manipulation" vulnerabilities were not patched as the solutions are beyond the scope of this paper. Future work could include an in-depth analysis of the AI model used for facial recognition and detection to prevent the model from accepting manipulated user input such as printed photos and sunglasses. Furthermore, tests could be performed on the AI model and devices to determine what made some of the photos cause the system to crash. Also, we plan to add a multimedia supplement walking through the exercises.

6. CONCLUSIONS

It has become increasingly crucial for Cybersecurity and CS students to possess the skills needed to understand the different components of a software application and how they relate to one another in the system. The process of Reverse Engineering can provide students with these skills. This paper provides an IoT-based framework to accomplish this goal, developing various labs to enhance students' competency on vulnerability analysis by Reverse Engineering.

7. REFERENCES

Ali, M. (2005). Why teach Reverse Engineering. In ACM SIGSOFT Software Engineering Notes (pp. 1–4). ACM Digital Library. https://doi.org/10.1145/1082983.1083004

Bellettini, C., Marchetto, A., & Trentini, A. (2004). WebUml: Reverse Engineering of web applications. In SAC '04: Proceedings of the 2004 ACM symposium on Applied computing (pp. 1662-1669). ACM Digital Library. https://doi.org/10.1145/967900.968231

Canfora, G., & Penta, M. (2007). New Frontiers of Reverse Engineering. In Future of Software Engineering, FOSE 2007 (pp. 326-341). IEEE Xplore. https://doi.org/10.1109/FOSE.2007.15

Canfora, G., Penta, M., & Cerulo, L. (2011). Achievements and challenges in software Reverse Engineering. In Communications of the ACM (pp. 142-151). ACM Digital Library. https://doi.org/10.1145/1924421.1924451

Chikofsky, E., & Cross J. (1990). Reverse engineering and design recovery: a taxonomy. In IEEE Software (pp. 13-17). IEEE Xplore. https://doi.org/10.1109/52.43044

Fezari, M. & Al Dahoud, A. (2018). Integrated Development Environment "IDE" For Arduino. Research Gate. https://www.researchgate.net/publication/328615543_Integrated_Development_Environment_IDE_For_Arduino

Li, Z., Chou, E., & McAllister, C. (2022). An IoT Based New Platform for Teaching Web Application Security. CYBERSECURITY PEDAGOGY & PRACTICE JOURNAL.

Li, Z., Ren, H., Chou, E., Liu, X., & McAllister, C. D. (2022). Retrieving Forensically Sound Evidence from the ESP Series of IoT Devices. IEEE Internet of Things Journal.

Lueth, K. L. (2015). IoT market analysis: Sizing the opportunity. IoT Analytic Report. March.

Morgan, S. (2020, Nov. 13). Cybercrime To Cost the World $10.5 Trillion Annually by 2025. In Cybercrime Magazine. https://cybersecurityventures.com/cybercrime-damages-6-trillion-by-2021/

Müller, H., Jahnke, J., Smith, D., Storey, M., Tilley, S., & Wong, K. (2000). Reverse engineering: a roadmap. In ICSE '00 (47-60). ACM Digital Library. https://doi.org/10.1145/336512.336526

Patel, R., Coenen, F., Martin, R., & Archer, L. (2007). Reverse Engineering of Web Applications: A Technical Review.

Potter, B. (2009). Microsoft SDL threat modelling tool. Network Security, 2009(1), 15-18.

Radware (2022). Global Threat Analysis Report. https://www.radware.com/getattachment/3d26f50b-f2a7-4ffa-9a84-1b5a598a0b27/2021-2022-Global-Threat-Analysis-Report_2022-FINAL-V2.pdf.aspx

Shwartz, O., Mathov, Y., Bohadana, M., Elovici, Y., & Oren, Y. (2018). Reverse Engineering IoT Devices: Effective Techniques and Methods. In IEEE Internet of Things Journal (pp. 4965-4976). IEEE Xplore. https://doi.org/10.1109/JIOT.2018.2875240

Taylor, C., & Collberg, C. (2016). A tool for teaching Reverse Engineering. In 2016 USENIX Workshop on Advances in Security Education, ASE 2016. USENIX. https://www.usenix.org/conference/ase16/workshop-program/presentation/taylor

Wikipedia contributors. (2022, July 5). ESP32. In Wikipedia, The Free Encyclopedia. Retrieved 19:04, July 15, 2022, from https://en.wikipedia.org/w/index.php?title=ESP32&oldid=1096637291

Appendix: Set up the lab environment

It is assumed that cybersecurity students participating in the course understand how to set-up a Kali Linux Virtual Machine using tools such as Virtual Box. The screenshots taken for this tutorial are on a Windows machine, but the entire curriculum could be performed on a Kali Linux VM.

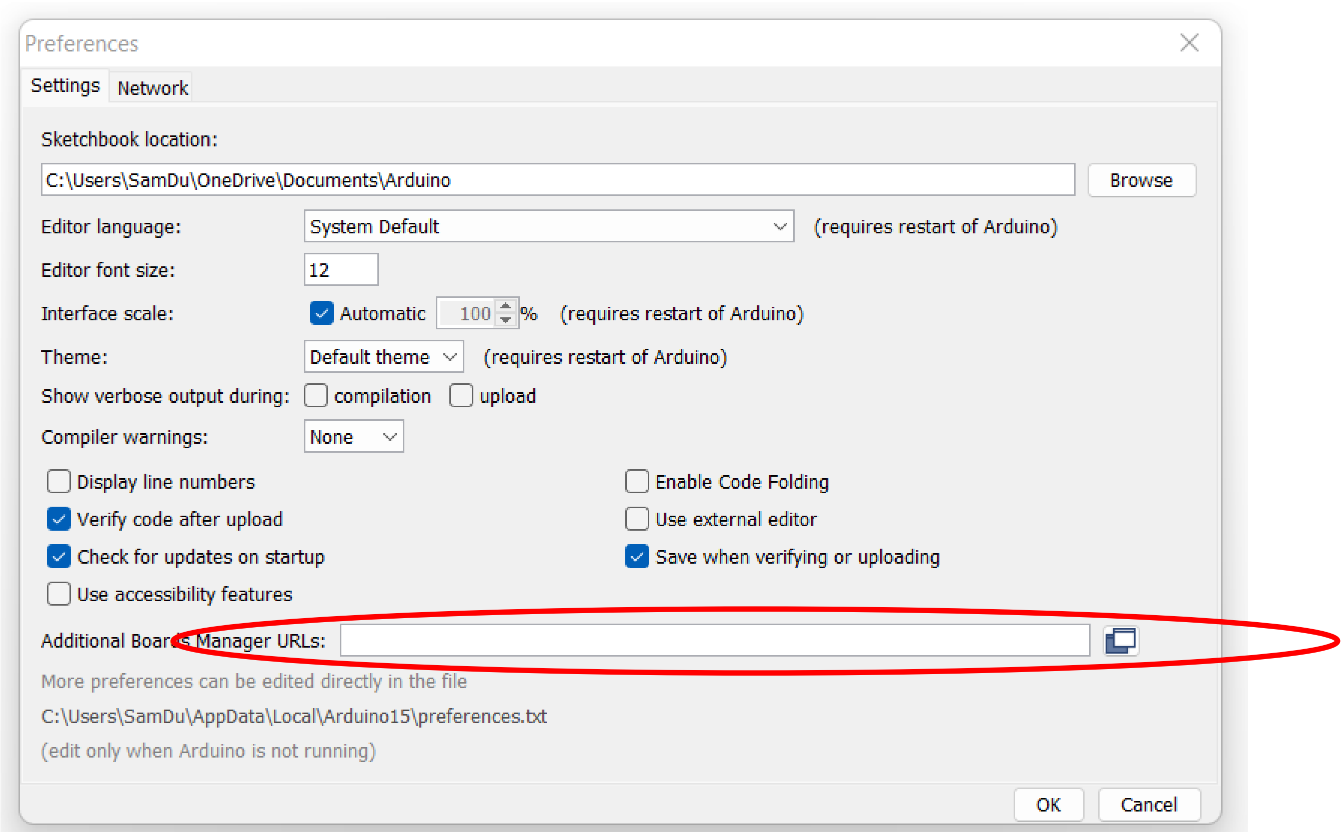
1. **Download the latest Arduino IDE**

    1. Go to https://www.arduino.cc/en/software
    2. Select the appropriate download option for your operating system. For the purposes of this paper and for simplicity, we recommend starting with the Windows operating system

    3. Once the installer has been downloaded, follow the installation instructions
    4. Open the Arduino IDE via the start menu or desktop shortcut

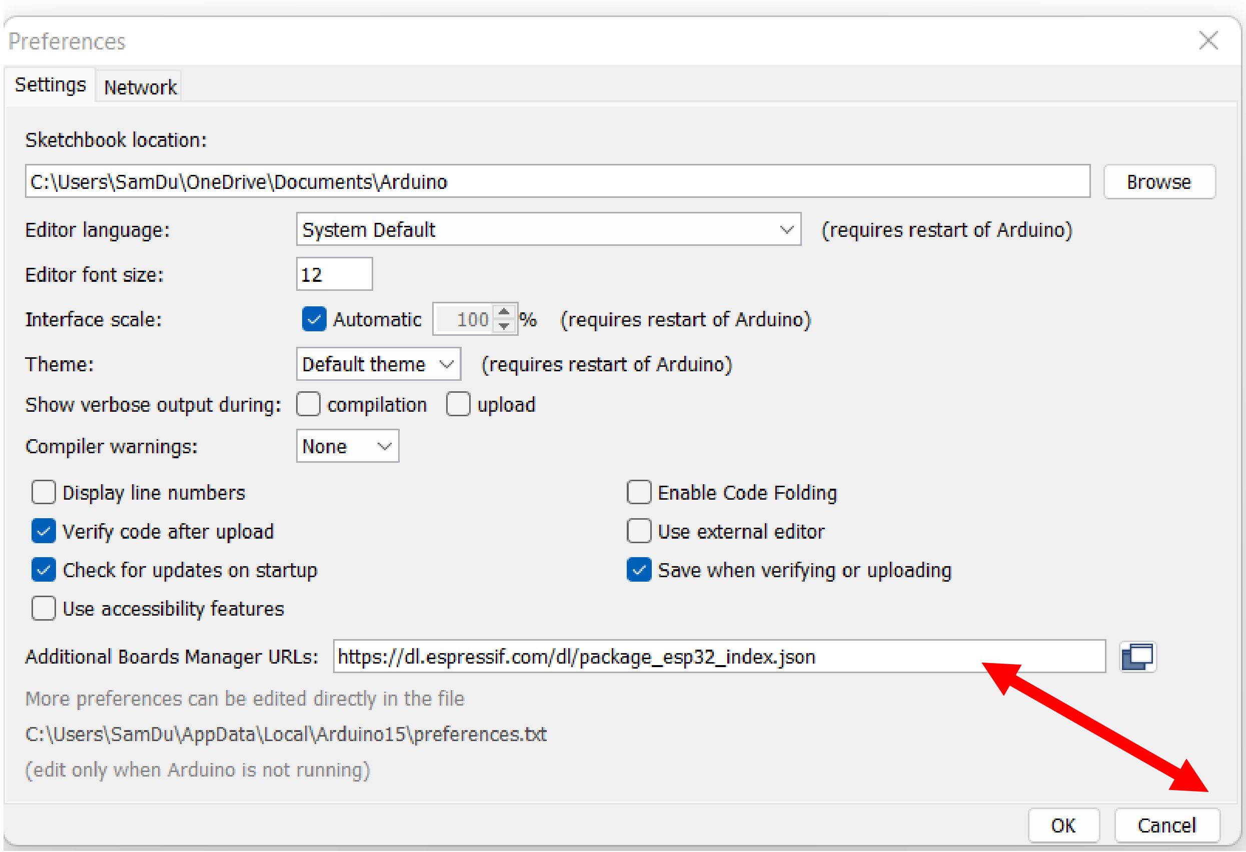
It should look like this:

5. Navigate to File->Preferences->AdditionalBoardsManagerURLS:

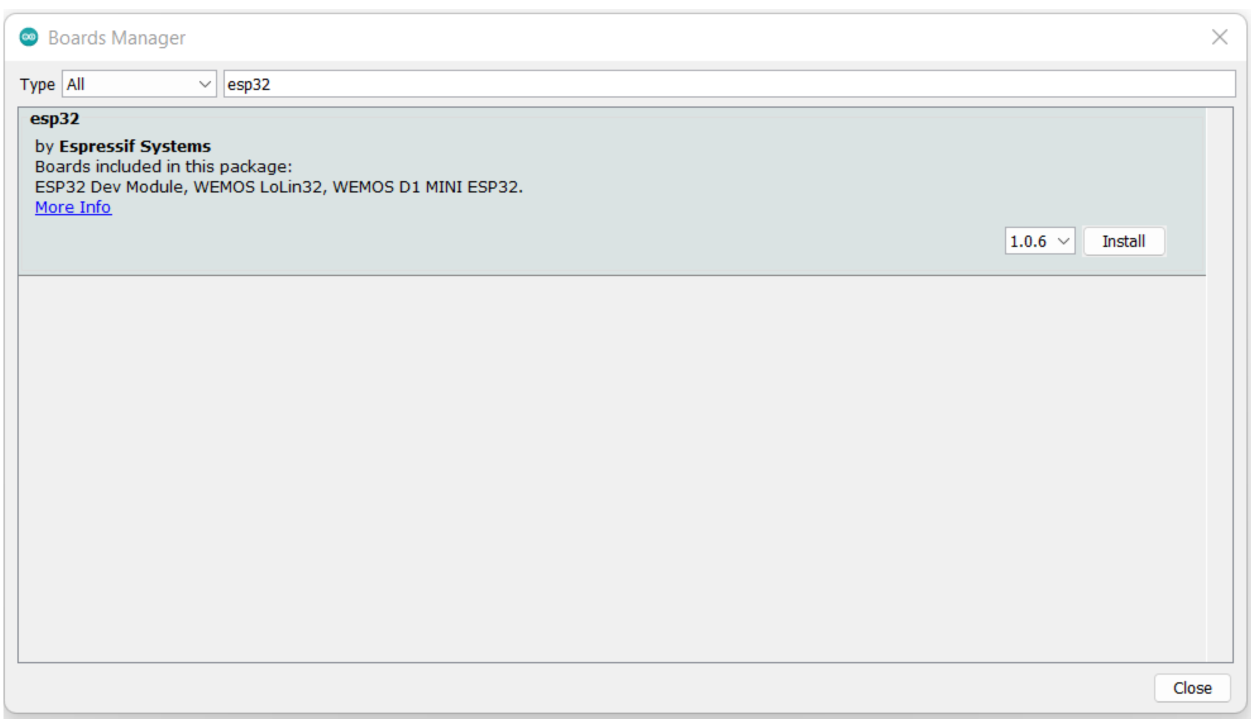


6. Paste this URL into the indicated textbox and click “ok”:

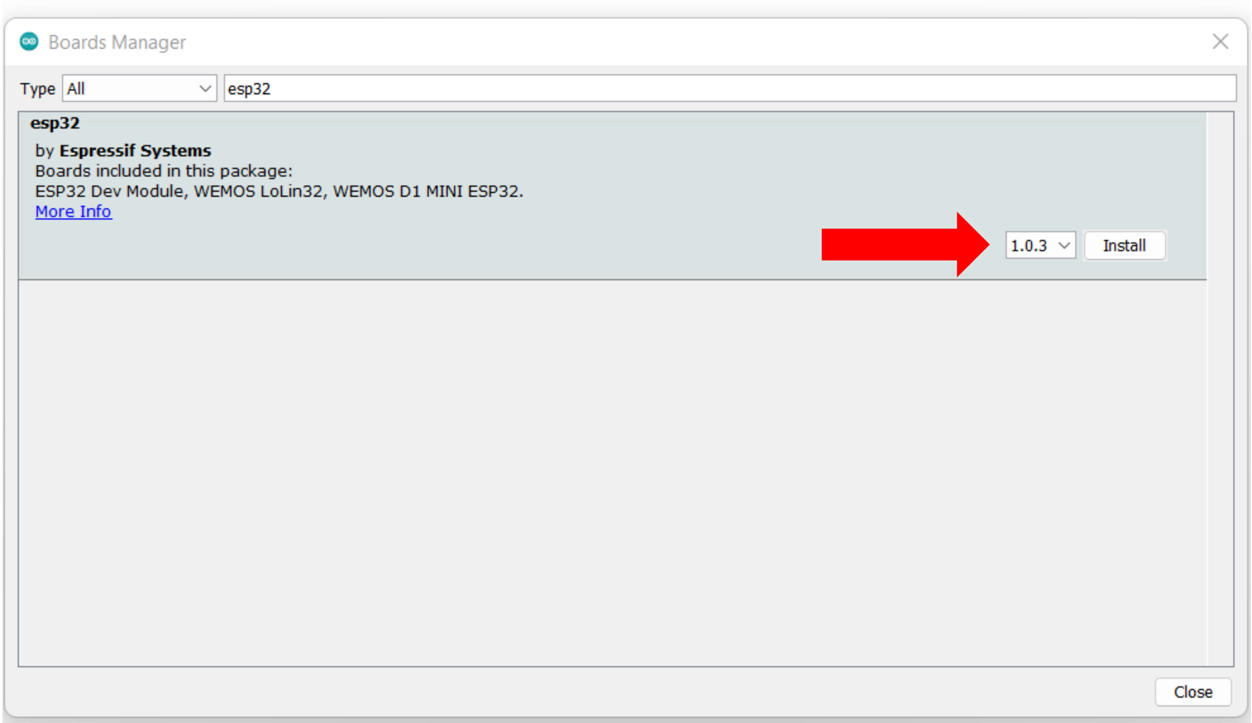
https://dl.espressif.com/dl/package_esp32_index.json



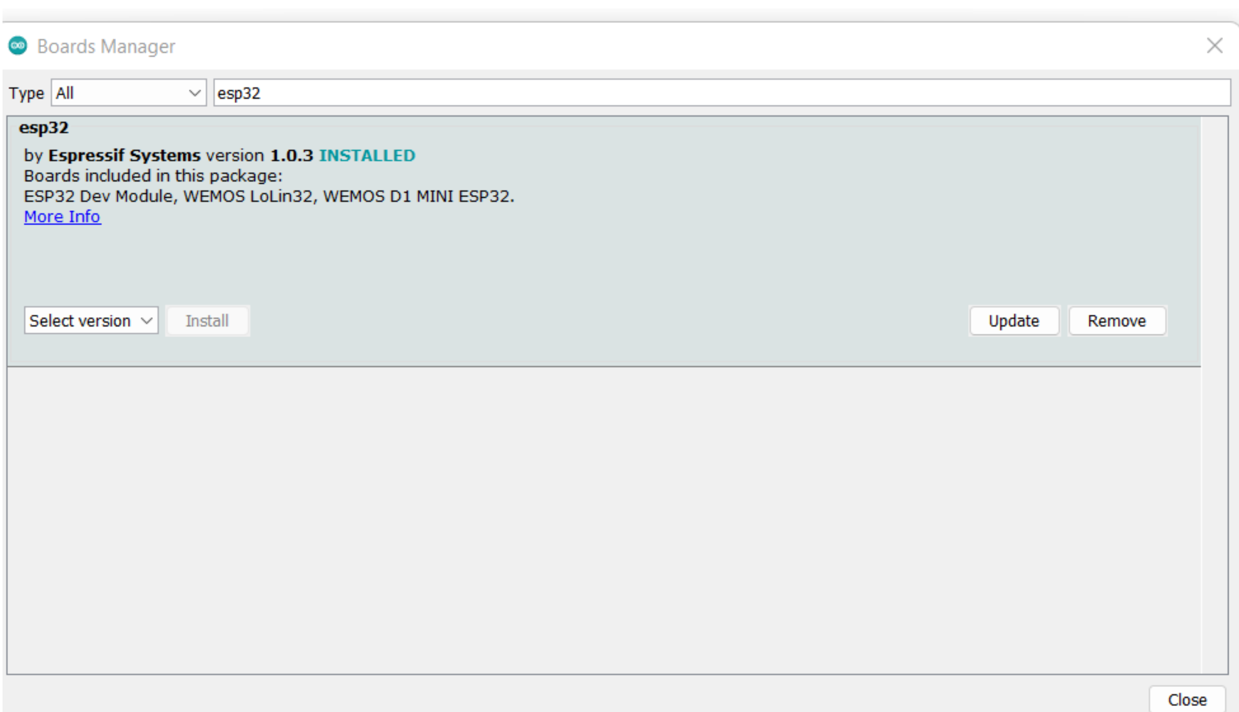
7. Navigate to tools->Board->Boards Manager-> and type "esp32" in the search bar. If the additional package was added in the previous step, you should see esp32 as an option:



8. In the right corner, select the 1.0.3 option and click install



9. Wait for the add-on to download. When it is finished you should see that it is installed:



10. Plug in the device to your computer via a USB cable. The cable must support data transportation, not just power.

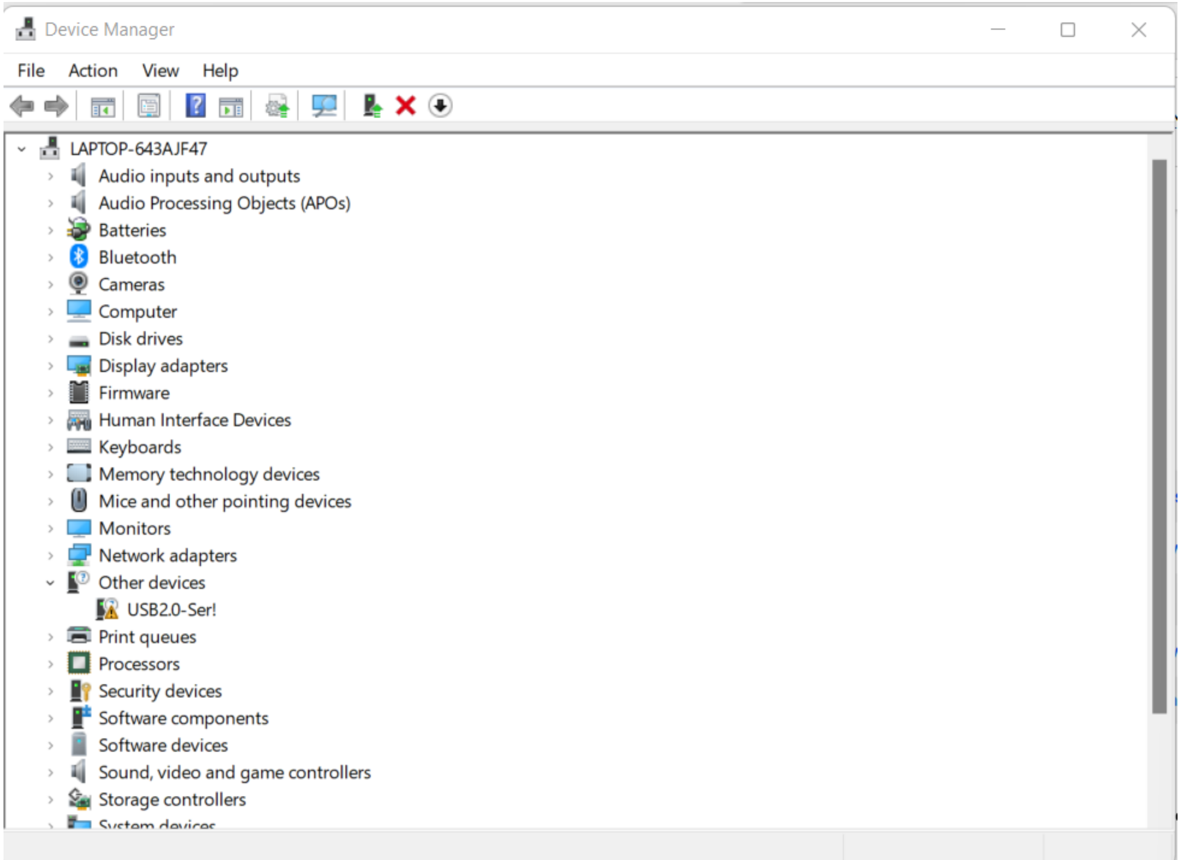
11. Navigate to Tools->Port. You should see a selected COM port available.

12. If "Port" is grayed out, then you will need to install the proper UART driver for the device. Download the driver available at this link: https://www.silabs.com/developers/usb-to-uart-bridge-vcp-drivers

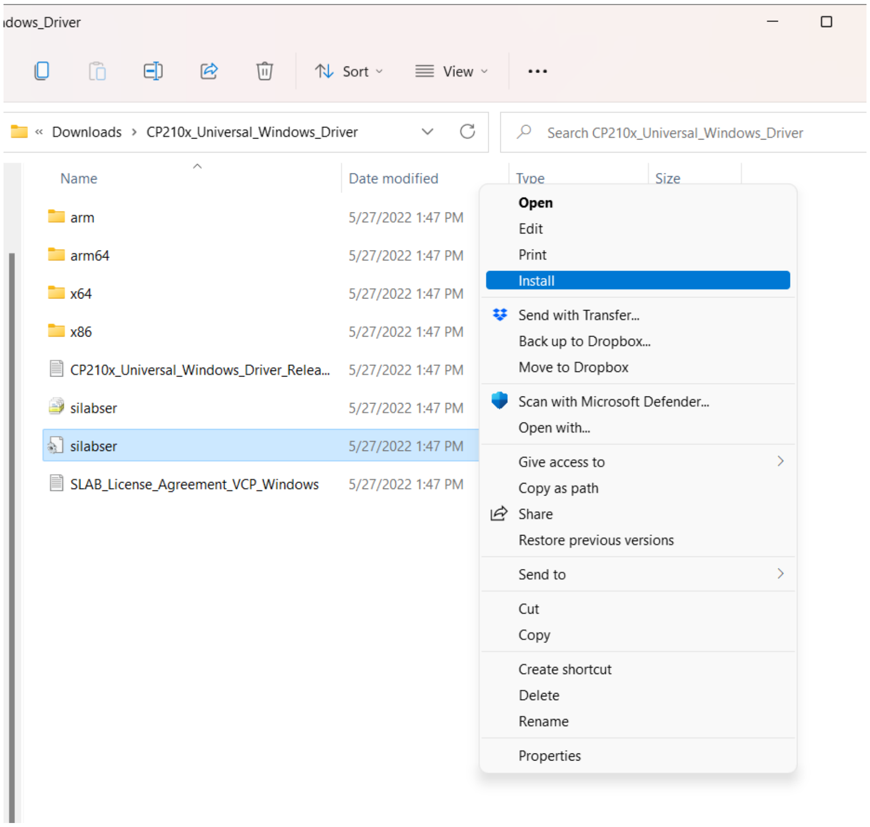
    1. Select the version compatible with your OS:

    2. Wait for the folder to download and extract the contents

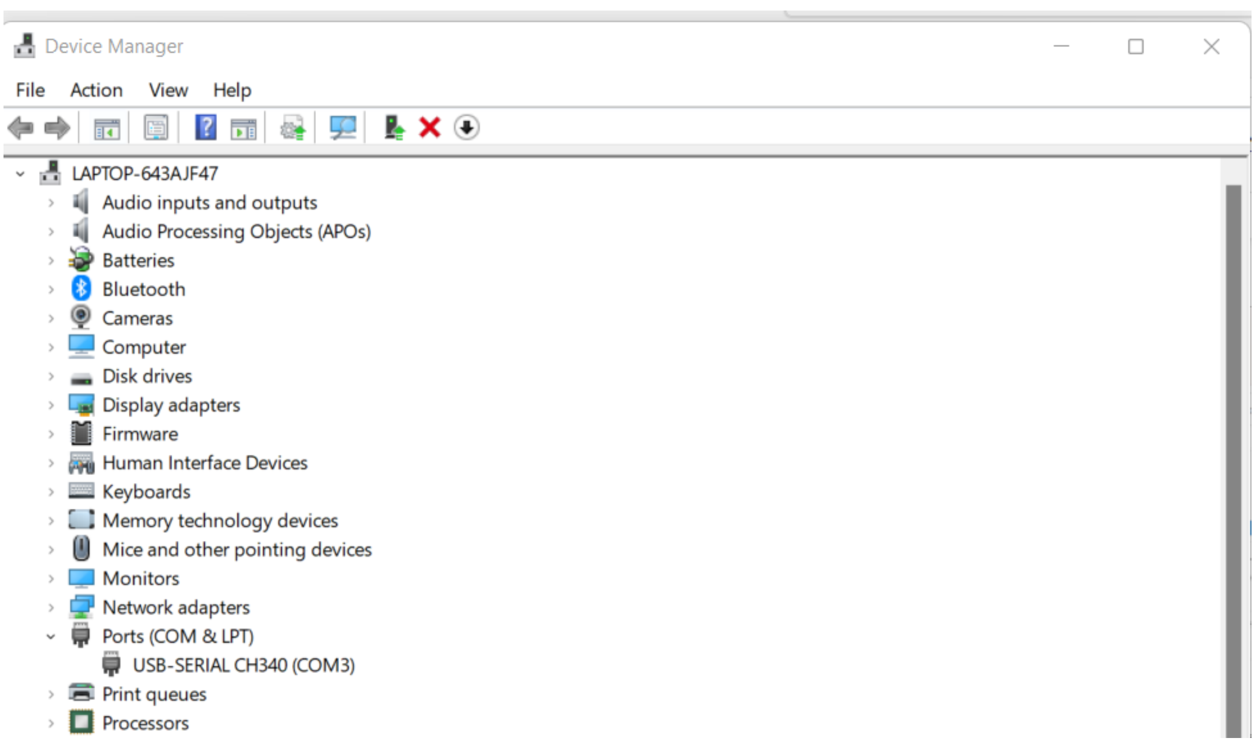
    3. Go to your start menu and type in device manager. Locate the device in the "Other devices" section. This indicates that the device is not being recognized by the proper driver



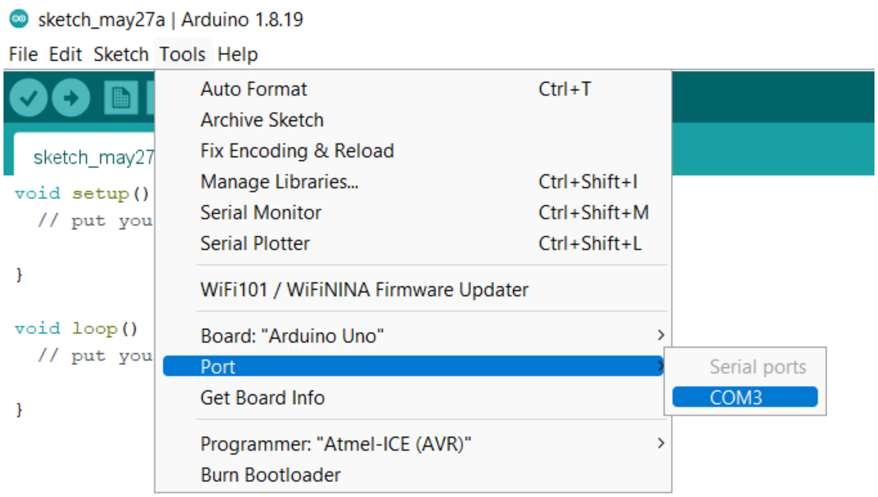
4. Navigate to the extracted driver folder, right click on the "silabser.inf" file and click install



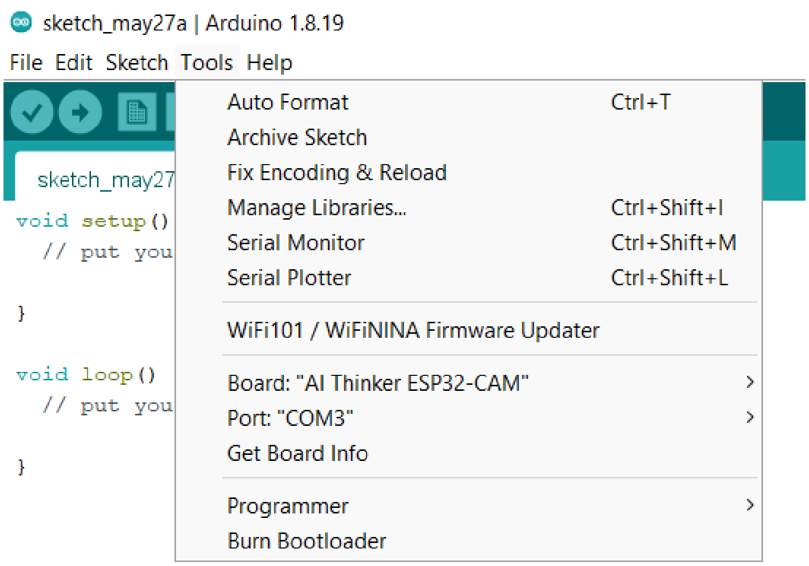
5. Wait for the install to complete, then re-open device manager. You should see the device listed under "Ports (COM & LPT)"



13. Once the proper driver is installed, re-open the Arduino IDE. Navigate to Tools -> Ports. Select the COM option available if it is not already selected.

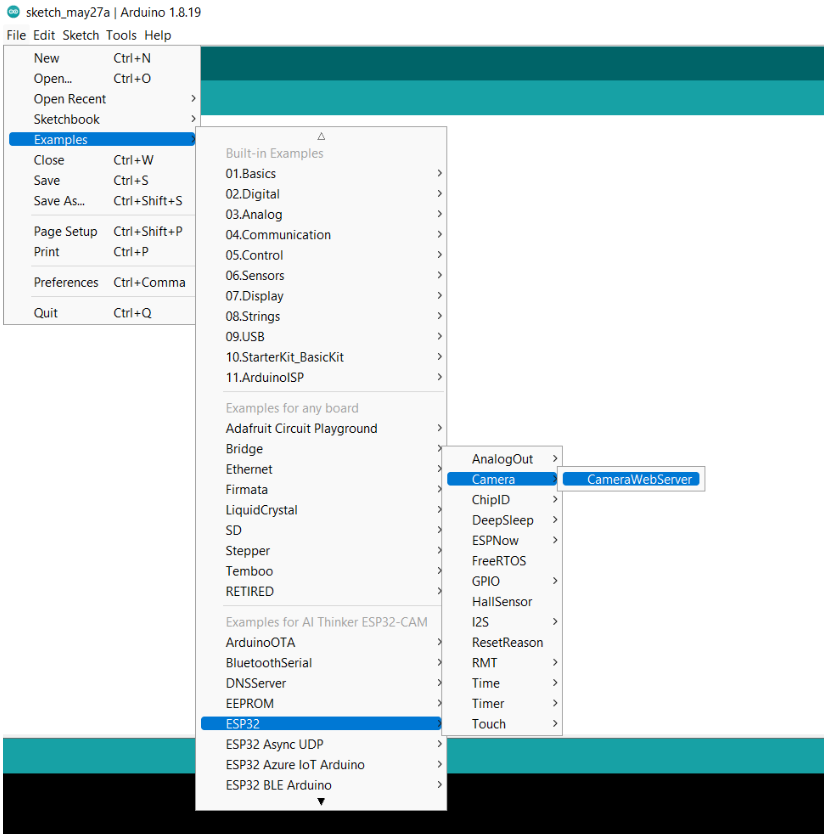


14. Then go to Tools -> Board -> ESP32 Arduino -> AI Thinker ESP32-CAM. The result should look like this (the COM port may be different):

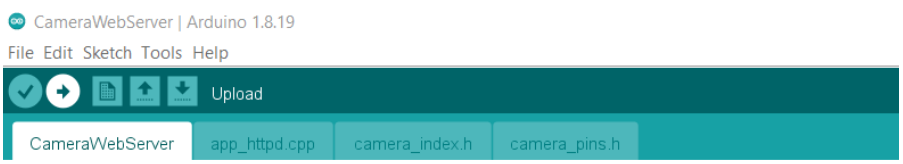


2. Find CameraWebServer Application and Flash to ESP32-CAM Device:

a. The Arduino IDE with the esp32 package installed comes with several example programs. Navigate to File -> Examples -> ESP32 -> Camera -> CameraWebServer.

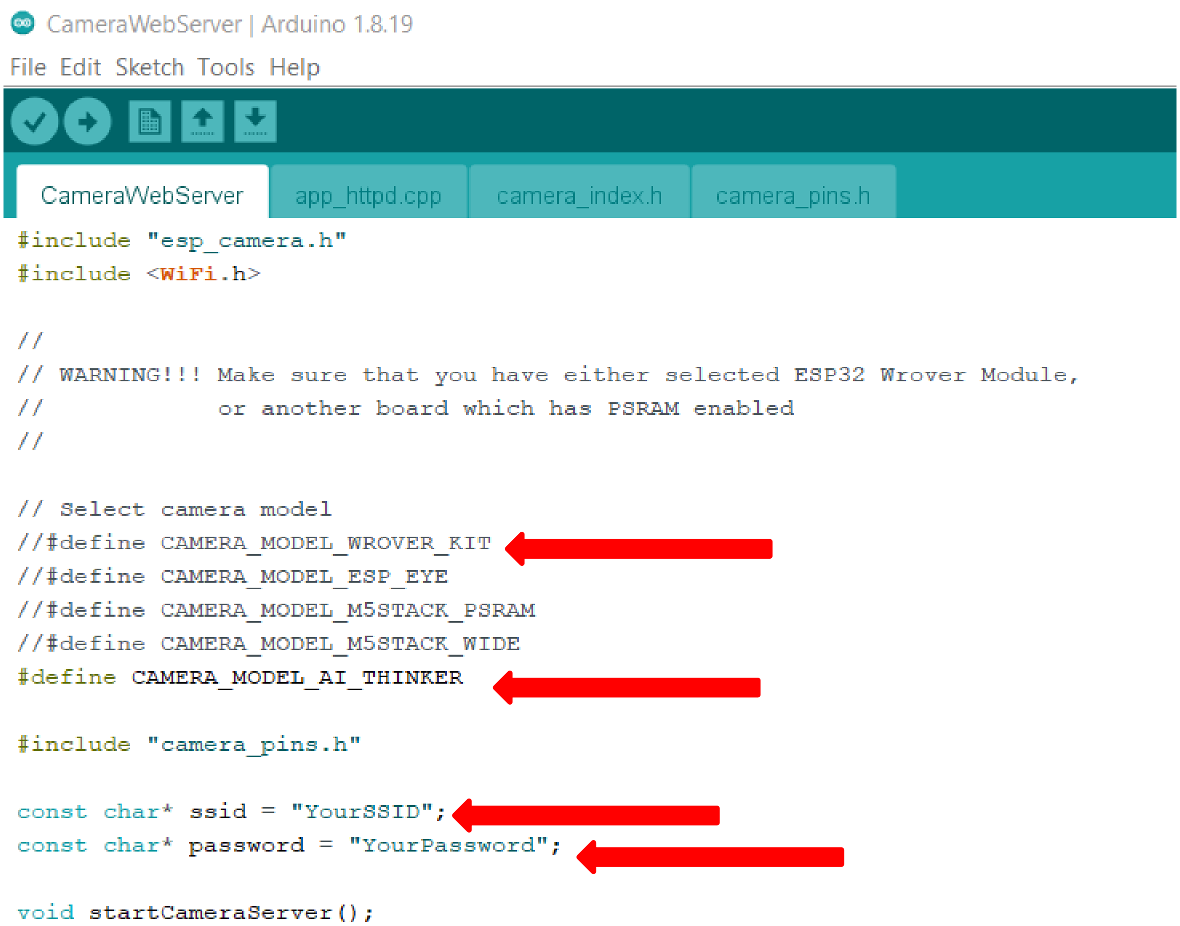


b. Load the program, it should look like this:



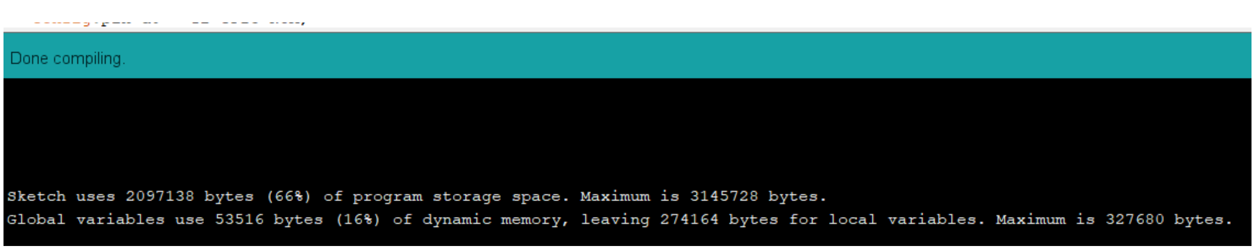

```
#include "esp_camera.h"
#include <WiFi.h>

//
// WARNING!!! Make sure that you have either selected ESP32 Wrover Module,
//            or another board which has PSRAM enabled
//

// Select camera model
#define CAMERA_MODEL_WROVER_KIT
//#define CAMERA_MODEL_ESP_EYE
//#define CAMERA_MODEL_M5STACK_PSRAM
//#define CAMERA_MODEL_M5STACK_WIDE
//#define CAMERA_MODEL_AI_THINKER

#include "camera_pins.h"

const char* ssid = "*********";
const char* password = "*********";

void startCameraServer();

void setup() {
```

c. Comment out the " define CAMERA_MODEL_WROVER_KIT" line, and uncomment the "#define CAMERA_MODEL_AI_THINKER". Replace the "ssid" and "password" variables with your choice of ssid and password. Your code should now look something like this:

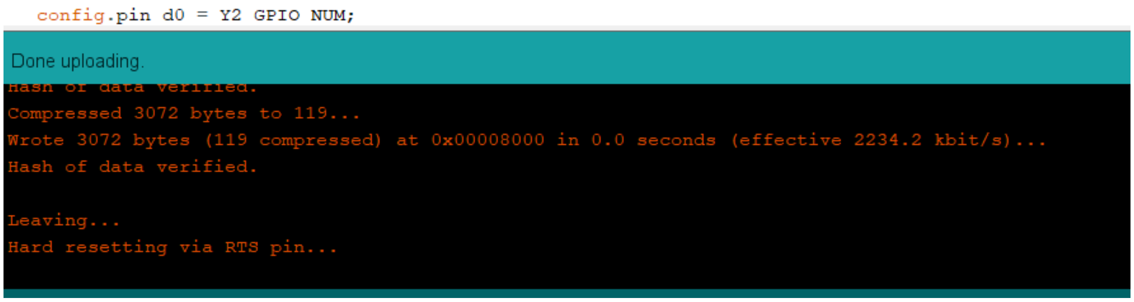


d. Click File -> Save to save the modifications. You may be prompted to save the sketch in a folder of your choosing.

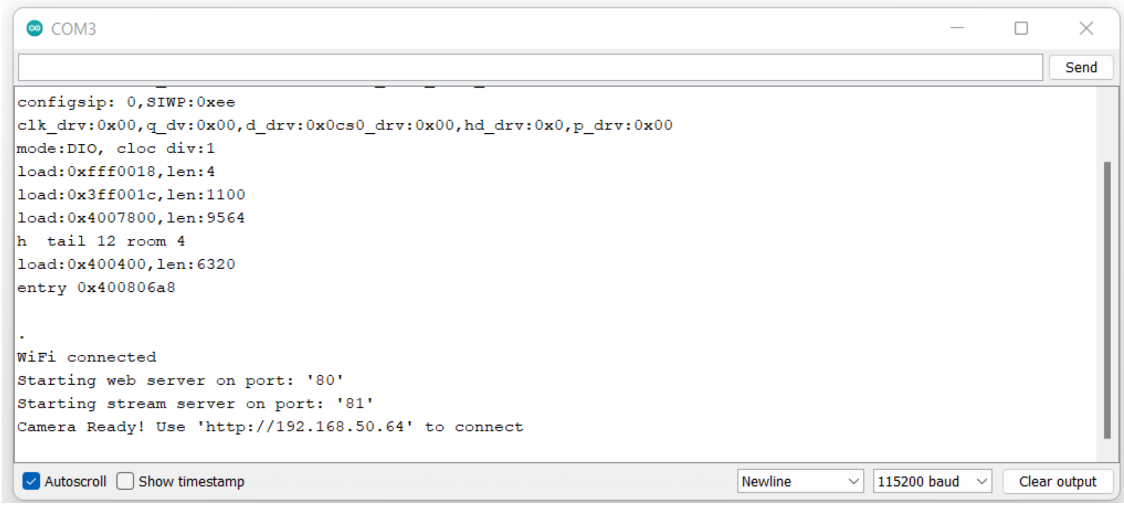
e. Click the check mark in the top left to compile the code. If it compiles successfully, the output at the bottom of the IDE should look something like this:



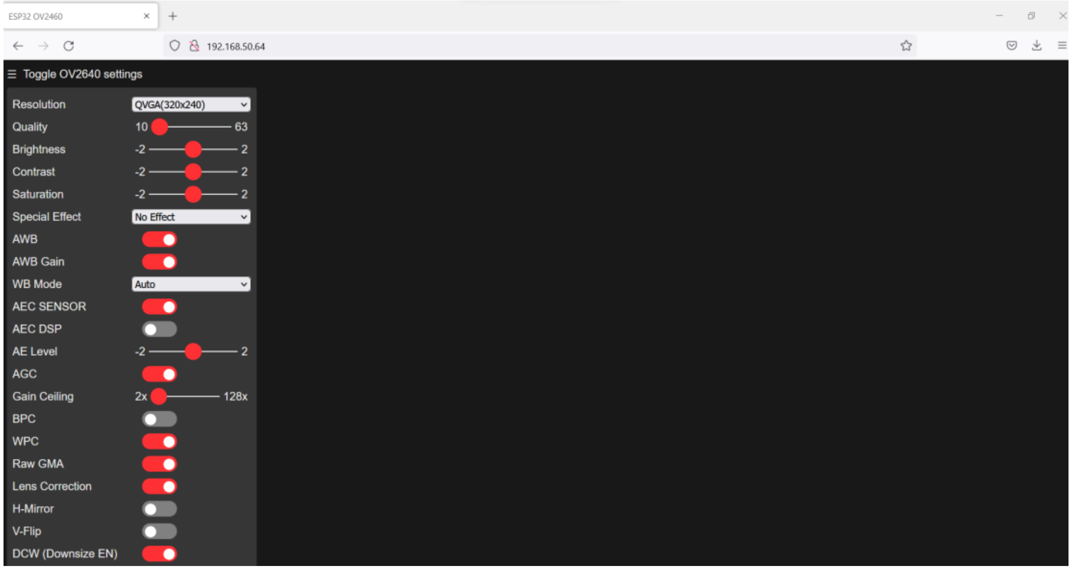
f. Next, click the arrow just to the right of the check mark to upload the code to the Device. Once again, make sure the proper COM port is selected and the proper device type is selected as well. If the code is uploaded properly, you should see something like this at the bottom of the IDE:



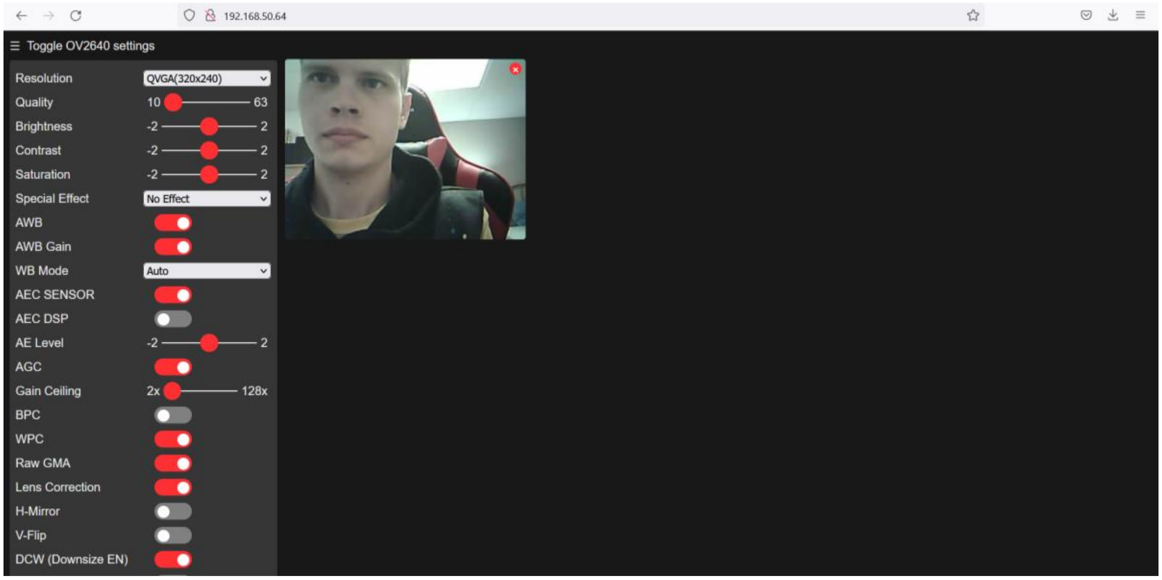
g. To access the IP address information for the web application, you must use the serial monitor. Click Tools -> Serial Monitor and set the baud rate to 115200 baud. Then click the RST button on the device. You should see something like this as an output:



h. Finally, use a device connected to the same Wi-Fi network that matches the SSID and password in the code. Open a web browser and type in the address IP displayed in the serial monitor. You should be directed to a webpage like this:

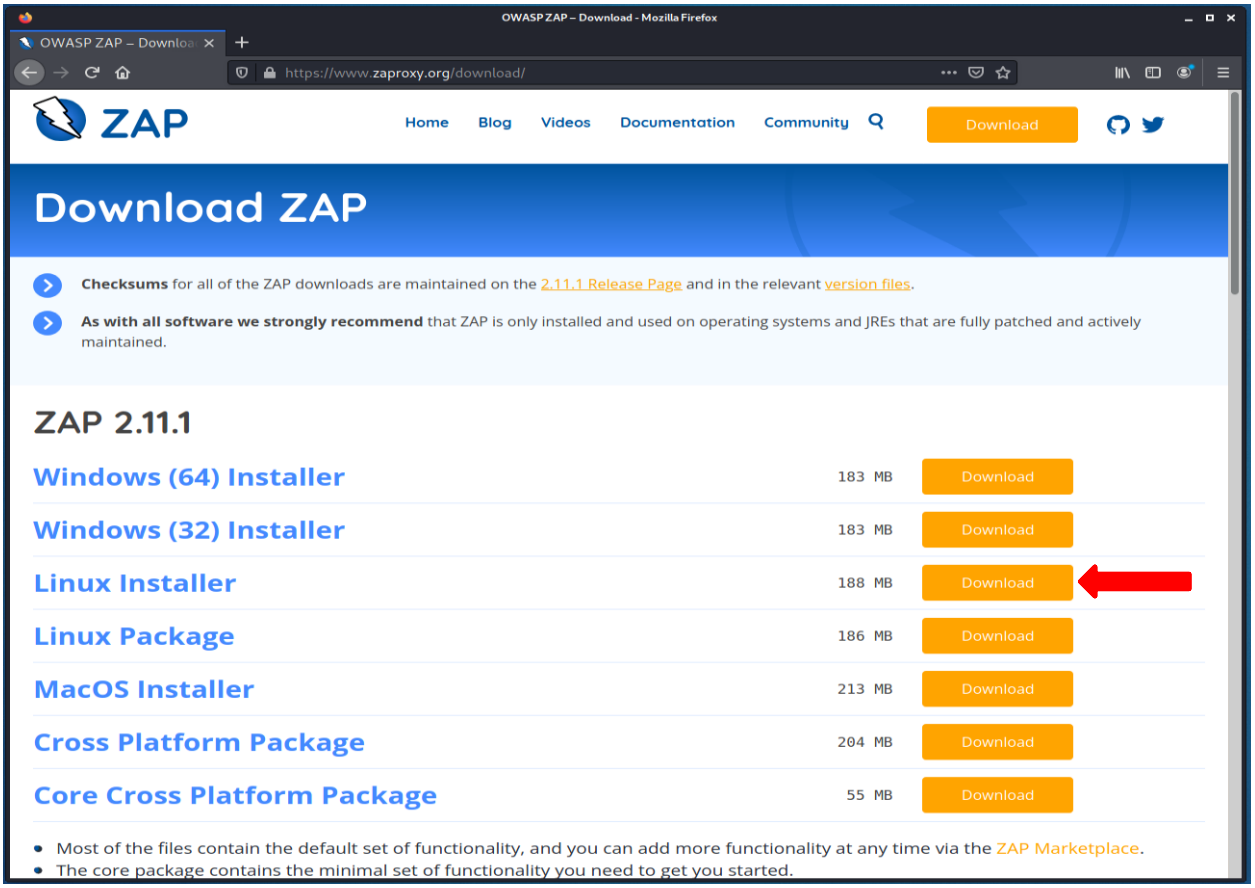


i. Toggle on the "Face Detection" and "Face Recognition" features and click "Start Stream". A video stream of the device's camera should appear. If this works correctly, you are ready to start the reverse and re-engineering process on the application!

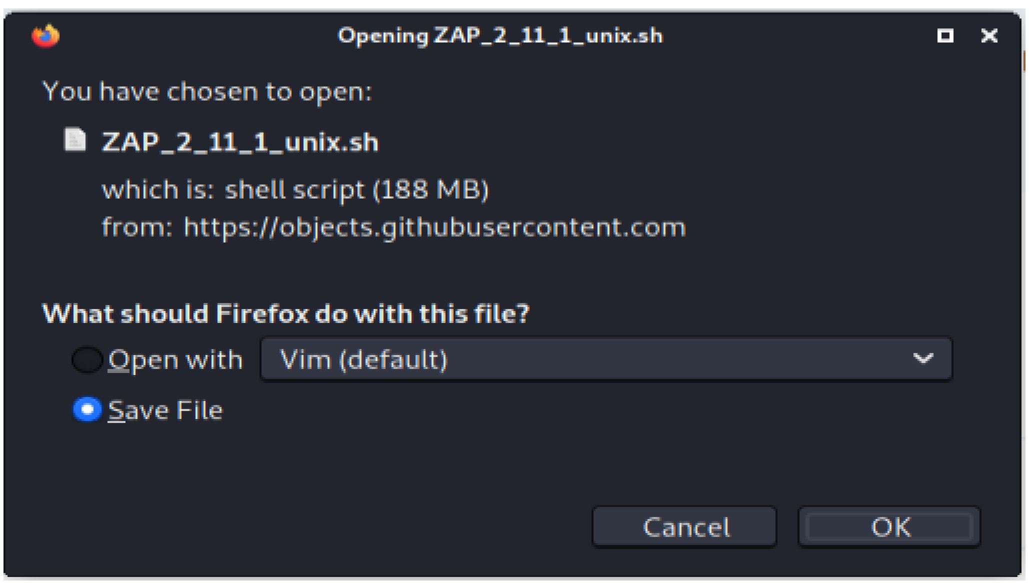


3. Install OWASP ZAP on Kali Linux VM

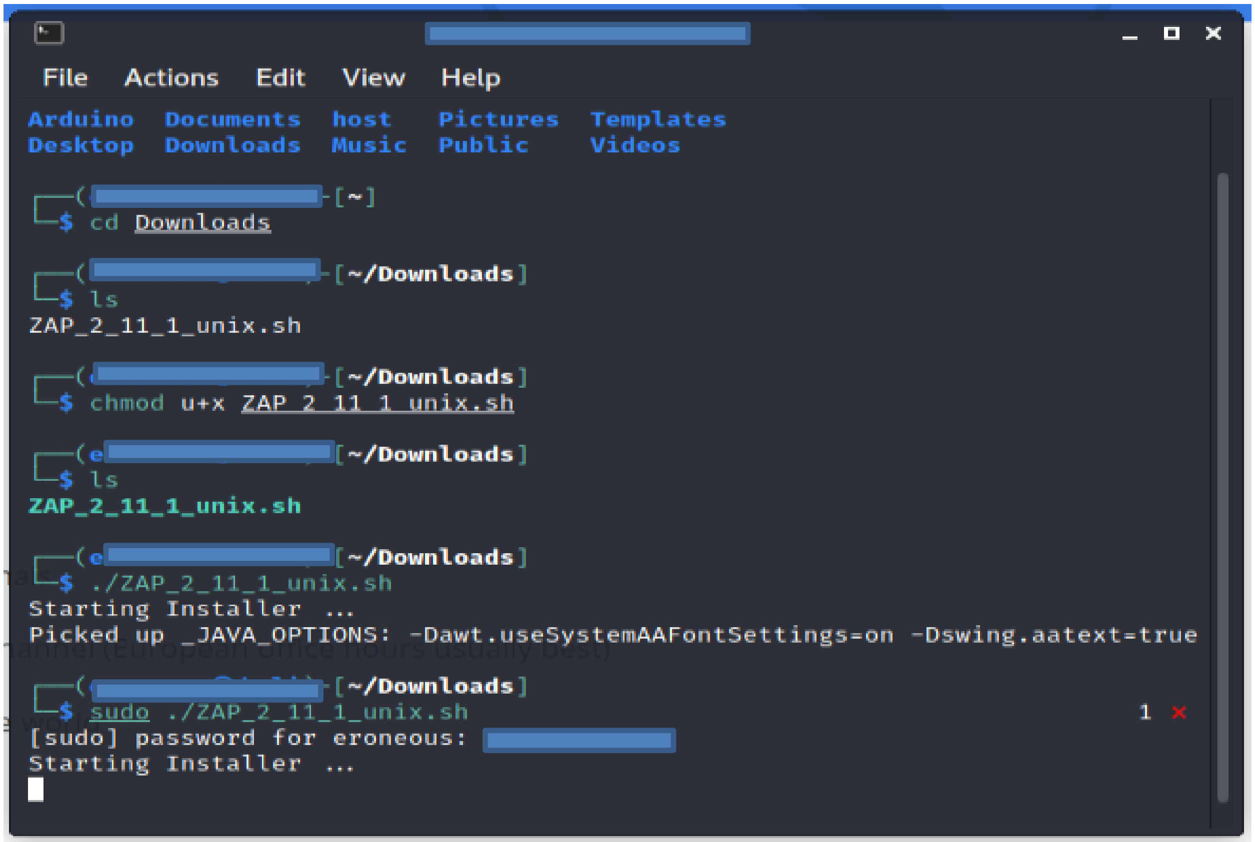
1. Start your Kali Linux VM. Visit this webpage: https://www.zaproxy.org/download/. Click the "Linux Installer" download option:



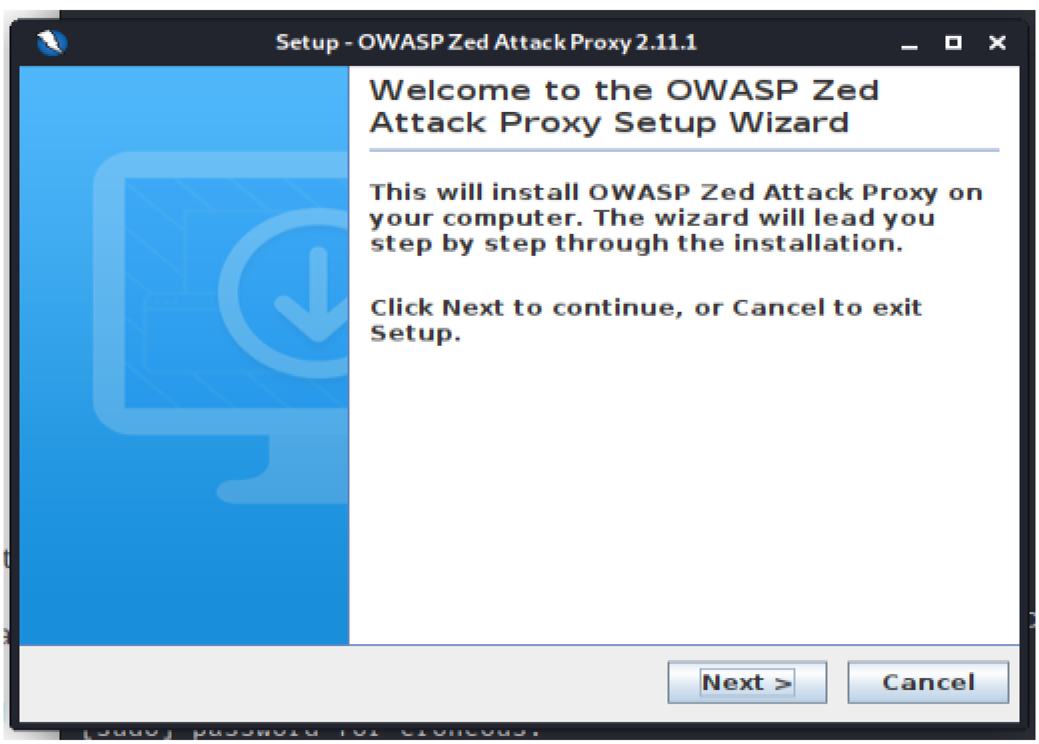
2. Save the file when prompted and click "OK"



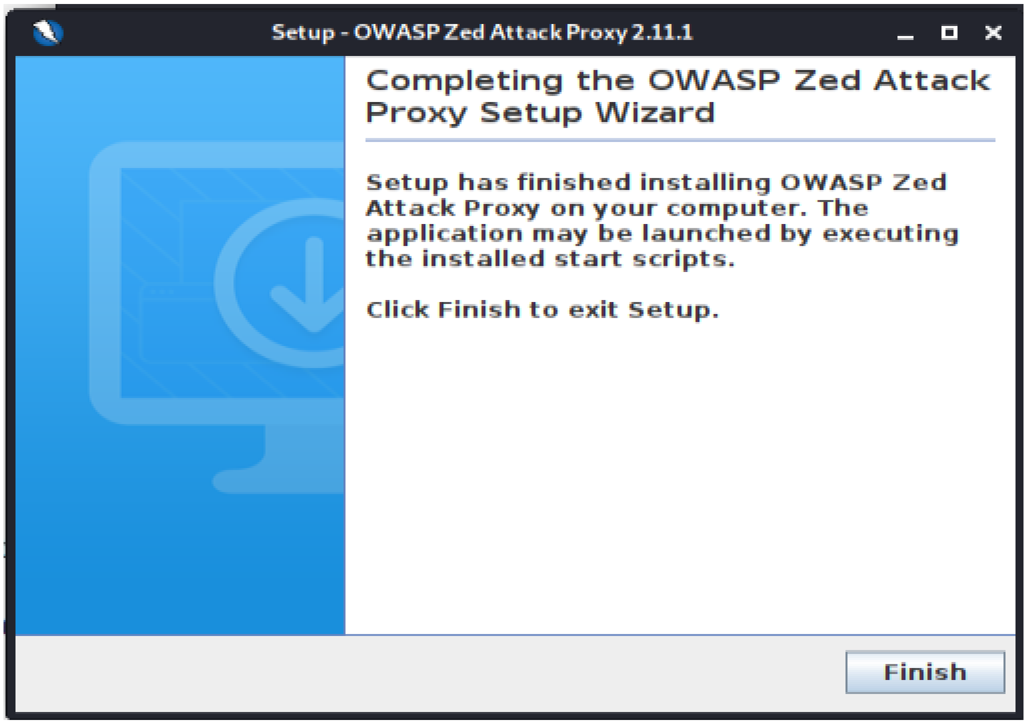
3. Open a terminal and navigate your downloads folder. Verify the "ZAP_2_11_1_unix.sh" file is there with he "ls" command. Enter "chmod u+x ZAP_2_11-1_unix.sh" to change the file to an executable. Then run the command "sudo ./ZAP_2_11_1_unix.sh" to run the installer:



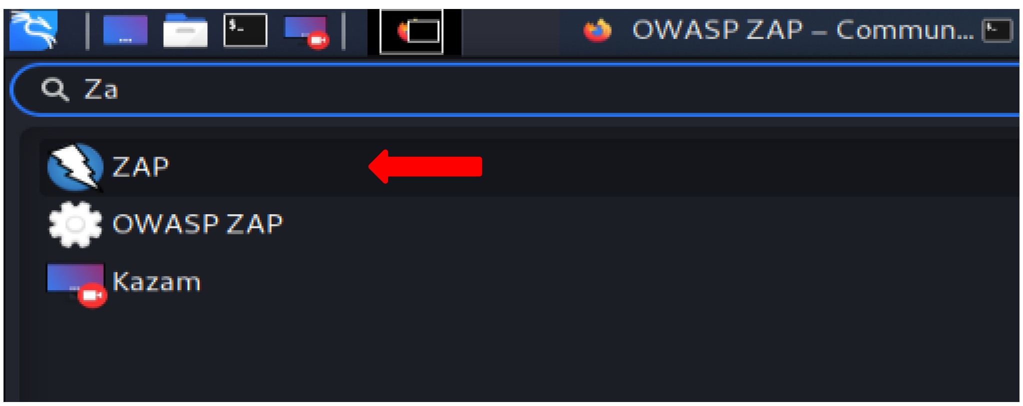
4. A dialog box will pop up, click next and agree to the terms of agreement:



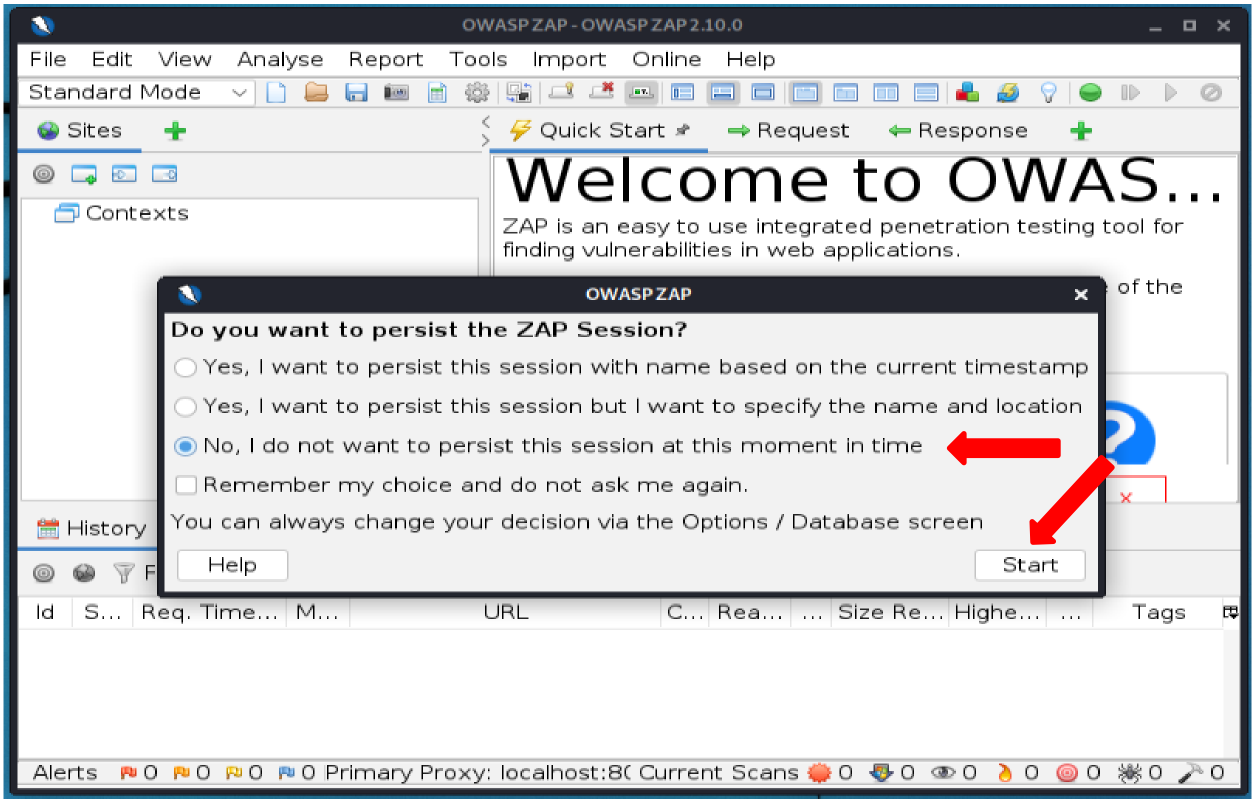
5. Click "Finish" to finish the installation:



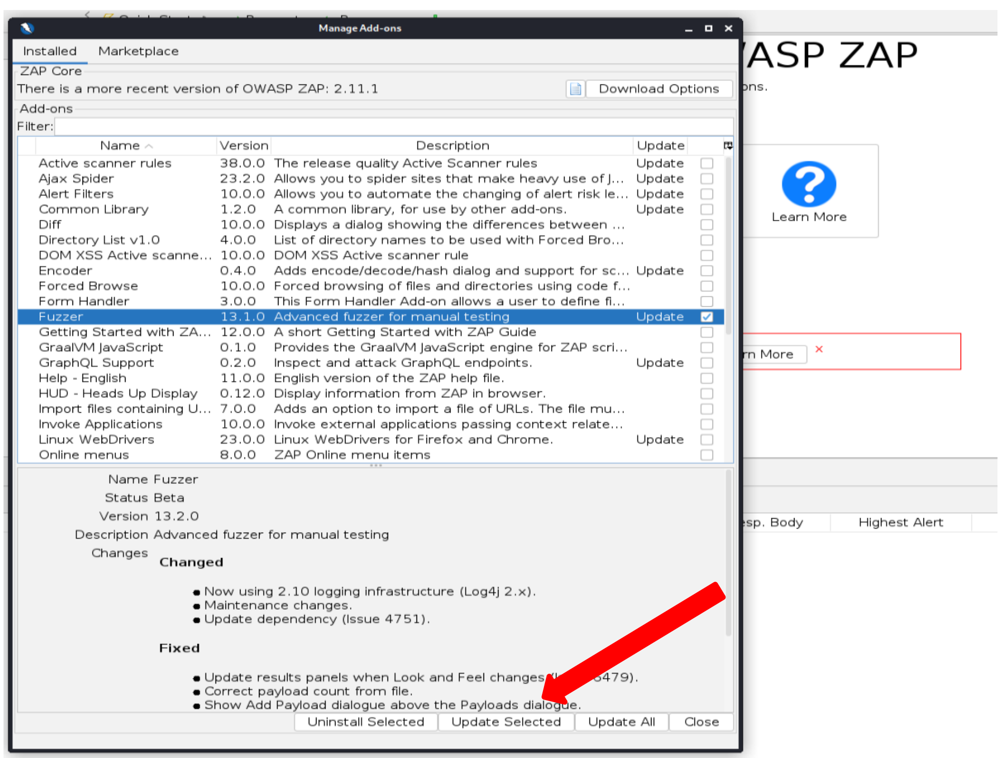
6. Click the menu in the top left corner and type Zap. You should see ZAP installed on your machine. Click the application to run it:



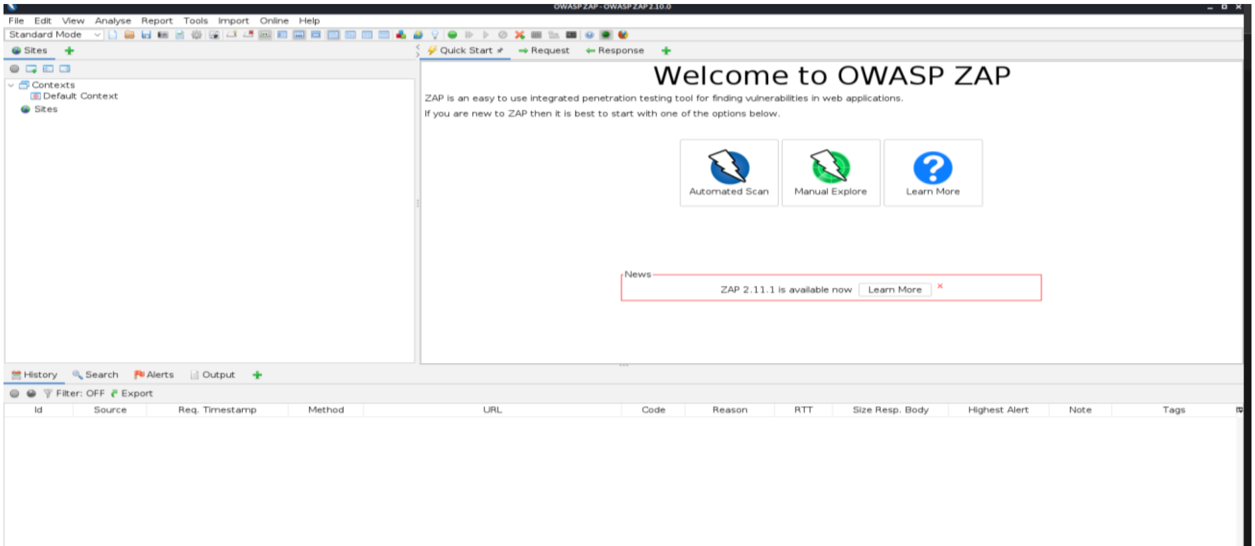
7. Select no for the persistent ZAP session and click start:



8. A manager add on box may pop up. If so, select the "Fuzzer" option and then click "Update Selected":



9. Now the OWASP ZAP tool should be ready to use, and it should look something like this:



10. With this installed, students should be read to follow the detailed instructions outline in the paper to practices the vulnerability detection via reverse engineering and re-engineering.