**Logo, company name

Description automatically generated**

Template based on the Centers for Medicare & Medicaid Services, Information Security & Privacy Management’s Assessment

**Security Assessment Report**

Version N.0

May 1, 2023

Table of Contents

[1. Summary 3](#_Toc127779445)

[1. Assessment Scope 3](#_Toc127779446)

[2. Summary of Findings 3](#_Toc127779447)

[3. Summary of Recommendations 4](#_Toc127779448)

[2. Goals, Findings, and Recommendations 4](#_Toc127779449)

[1. Assessment Goals 4](#_Toc127779450)

[2. Detailed Findings 5](#_Toc127779451)

[3. Recommendations 5](#_Toc127779452)

[3. Methodology for the Security Control Assessment 5](#_Toc127779453)

[4. Figures and Code 7](#_Toc127779454)

[4.1.1 Process flow of System (this one just describes the process for requesting) 7](#_Toc127779455)

[4.1.2 Other figure of code 7](#_Toc127779456)

[5. Works Cited 7](#_Toc127779457)

# Summary

The overall goal of the security assemsment is to address and resovle and bugs or issues within the code that may lead to a security threat or risk at a later time or pose a threat currently. The findings of the assessment were that the code had many areas of weakness that could have left it susseptable to an attack if someone wished to do harm. The methods of discovery used were overall analysis of the code using things such as a SWAT analysis as well as a risk assessment. Throught these tools of discoverer it was discovered that the code was lacking in many common areas that other unsecure code usually fails to address. These areas include things like a failure to execute proper bounds checking, a failiure to use smart pointers and improper access control, among other. The recommendations are straight forward as they are the result of such common errors. The first is to make sure the code has secure bounds checking when loading in inputs into arrays to ensure the proper size. The second would be the use of smart pointers to prevent memory leaks. The last major recommendation would be the use of access control to make sure that only those who are authorized to be accessing the code are able to view it.

## Assessment Scope

The scope of the assessment include the many tools that were used in the creation of the code. The main application that the program was running of was the Arduino IDE used to execute code and upload it to edge devices. In this specific application the code was to be used on an Arduino BLE Sense 33. Other applications used include the TensorFlow library in unison with Google Colab where the predictive model was created and trained to be later condenced and transferred to the Arduino. The main programming language being used was C++, this left the most vulnerability. Python was also used to train and create the model but at the time of the assessment did not show any vulnerabilities.

## Summary of Findings

The finding discovered that there was a lack of smart pointers which was leaving an opportunity for a memory leak. There was also nothing in place the check if the inputs of the predictive model, which were given as C++ arrays, were of the proper size. The overall storage of the project was all public which also left the possibility for the TensorFlow model itself to be accessed incluing its specific weights and biases that were being stored. Of the findings discovered during our assessment, 0 were considered High risks, 2 Moderate risks, 2 Low, and 0 Informational risks. The SWOT used for planning the assessment are broken down as shown in Figure 2.

Figure 1. Findings by Risk Level

It was found that ther were two risks that could be considered moderate as they posed and threat to the integrity of the project. Those risks were the use of regular pointers and the failure to perform proper bounds checking.

Diagram

Description automatically generated

Figure 2. SWOT

Explain which issues were used from above SWOT (which are addressed in this assessment).

## Summary of Recommendations

The recommended corrective action is to start with replacing the regular C++ pointers with smart pointers. Then, to ensure that data being input as the input tensor and used to test the model is of the proper size, bounds checking should be implemented. This is as simple as making sure the input tensor being used matches the expected size of the original input tensor the model was trained on. The final recommendation would be to store the actual code in an orgazation to allow for the access control to be manipulated and secure the model in the process.

# Goals, Findings, and Recommendations

## Assessment Goals

The purpose of this assessment was to do the following:

* Ensure that the system was in compliance with regulations you had to deal with or any other requirements (to include the assignments themselves).
* Determine if the application was securely maintained.
* Etc.

## Detailed Findings

Vulnerabilties:

#1: In the original code rawpointers were used throughout, this left it open to the possibility of a memory leak if those pointers where not being properly deleted.

#2: In the original code there was not any cases of bounds checking. This is a vulnerability when it comes to the way the input data was being intered into the input tensor in the code. It was being run through a loop one element at a time which leaves the possibility data being copied into unintended locations if the input array was too large. This could lead to many problems possibly immeditly and most deffenitly down the line.

#3: The original code was being stored in a normal GitHub repository, this leaves it open to whomever may access it at any time. This can be a problem when the data being stored is sensitive and not something that should have public access.

#4 The original code had a hardcoded size being used for the tensor arena. Depending on the size of the model this arena size could be quiet large. This leaves the possibility for a buffer overflow if the given array is very large and there are other arrays being stored of similar size.

## Recommendations

Here’s where your fixes go (ensure you reference Table 2 for your ease of fix evaluation and explain why it matches that category)

Fixes:

#1: This fix will relate directly to vulnerability #1 from above. When it comes to the use of raw pointers, the fix is to use smart pointers. Smart pointers are created and destroyed dynamically which prevents memory leaks from happening in the case that a raw pointer is not properly deleted. As it relates to the Table 2 below, this fix would be considered easy. Simply switching out the pointers was all that needed to be done.

#2: As for the case of vulnerability #2, to properly check the bounds of the array it is very easy to simply check that the array is the same sime size of the input tensor being used earlier in the code. In doing so this ensures that there will not be a buffer overflow or any issues related to the insertion of an improperly sized array. As it relates to Table 2 below, this fix would be considered easy. A simple conditional before the loop to insert the array is called would easily solve the issue.

#3: To fix the access control of the code it is easiest to move it to an organization, this will allow for the code to be privatized and only desired users to be included to access. As it related to Table 2, this fix would be considered easy. Moving the code to an organization is straight forward and takes limited time and effort.

#4 To fix the tensor arena size problem we would just need to allocate it dynamically allowing for it to be deleted after the inference is completed. As it related to Table 2, this fix would be considered easy. The hardcoded tensor arena would have to be switched to a dynamic one and then deleated later in the code. There are no major complications in doing so.

# Methodology for the Security Control Assessment

**3.1.1 Risk Level Assessment (3.1.1)**

Each Business Risk has been assigned a Risk Level value of High, Moderate, or Low. The rating is, in actuality, an assessment of the priority with which each Business Risk will be viewed. The definitions in Table 1 apply to risk level assessment values (based on probability and severity of risk). While Table 2 describes the estimation values used for a risk’s “ease-of-fix”.

Table 1 - Risk Values

| Rating | Definition of Risk Rating |
| --- | --- |
| High Risk | Exploitation of the technical or procedural vulnerability will cause substantial harm to the business processes. Significant political, financial, and legal damage is likely to result |
| Moderate Risk | Exploitation of the technical or procedural vulnerability will significantly impact the confidentiality, integrity and/or availability of the system, or data. Exploitation of the vulnerability may cause moderate financial loss or public embarrassment to organization. |
| Low Risk | Exploitation of the technical or procedural vulnerability will cause minimal impact to operations. The confidentiality, integrity and availability of sensitive information are not at risk of compromise. Exploitation of the vulnerability may cause slight financial loss or public embarrassment |
| Informational | An “Informational” finding, is a risk that has been identified during this assessment which is reassigned to another Major Application (MA) or General Support System (GSS). As these already exist or are handled by a different department, the informational finding will simply be noted as it is not the responsibility of this group to create a Corrective Action Plan. |
| Observations | An observation risk will need to be “watched” as it may arise as a result of various changes raising it to a higher risk category. However, until and unless the change happens it remains a low risk. |

Table 2 - Ease of Fix Definitions

| Rating | Definition of Risk Rating |
| --- | --- |
| Easy | The corrective action(s) can be completed quickly with minimal resources, and without causing disruption to the system or data |
| Moderately Difficult | Remediation efforts will likely cause a noticeable service disruption   * A vendor patch or major configuration change may be required to close the vulnerability * An upgrade to a different version of the software may be required to address the impact severity * The system may require a reconfiguration to mitigate the threat exposure * Corrective action may require construction or significant alterations to the manner in which business is undertaken |
| Very Difficult | The high risk of substantial service disruption makes it impractical to complete the corrective action for mission critical systems without careful scheduling   * An obscure, hard-to-find vendor patch may be required to close the vulnerability * Significant, time-consuming configuration changes may be required to address the threat exposure or impact severity * Corrective action requires major construction or redesign of an entire business process |
| No Known Fix | No known solution to the problem currently exists. The Risk may require the Business Owner to:   * Discontinue use of the software or protocol * Isolate the information system within the enterprise, thereby eliminating reliance on the system   In some cases, the vulnerability is due to a design-level flaw that cannot be resolved through the application of vendor patches or the reconfiguration of the system. If the system is critical and must be used to support on-going business functions, no less than quarterly monitoring shall be conducted by the Business Owner, and reviewed by IS Management, to validate that security incidents have not occurred |

**3.1.2 Tests and Analyses**

To evaluate the effectiveness of our proposed solution, we conducted a series of tests and analyses using both synthetic and real-word data gathered from the original MNIST training data.

**3.1.2.1 Synthetic Data Tests:**

First, we generated synthetic data That mimics the dimensions of the original input tensor. We then applied our proposed solution to this synthetic data and compared its performance with that of existing methods. The results of our tests showed that our proposed solution did not accept the input data that was too large and accepted only the data of the proper size.

**3.1.2.2 Real-world Data Tests:**

To further validate our proposed solution, we conducted tests on real-world data collected from the MNIST data set input tensor data. We gathered a selection of 10 input tensors, each representing one of the numbers 0 – 9 and used them to test the model. The effectiveness of the model matched that of the model before the added security features were implemented.

**3.1.3 Tools**

This was completed using:

* Python: Python was used as the primary programming language for data cleaning, analysis, and modeling.
* Jupyter Notebook: Jupyter Notebook was used as an interactive development environment (IDE) to write and execute Python code.
* Pandas: Pandas is a Python library used for data manipulation and analysis. It was used extensively for data cleaning and transformation.
* Scikit-learn: Scikit-learn is a popular Python library used for machine learning tasks such as regression, classification, and clustering. It was used for building and evaluating the machine learning models.
* Matplotlib and Seaborn: Matplotlib and Seaborn are Python libraries used for data visualization. They were used to visualize the data and the results of the machine learning models.
* Git and GitHub: Git is a version control system used for managing code changes. GitHub is an online repository used to host the project code and collaborate with other team members.
* Google Colab: Google Colab is a cloud-based platform used for running Jupyter Notebooks. It was used to train the machine learning models on a GPU without the need for expensive hardware.

# Figures and Code

### Process or Data flow of System

Diagram

Description automatically generated

### Other figure of code

Recommendation #2 (Bounds checking)

Code Figure 1

Graphical user interface, text, application

Description automatically generated

4.1.2 Recommendation #4 (Dynamic tensor arena)

Code Figure 2

Graphical user interface, text

Description automatically generated

Recommendation #1 (Smart Pointers)

Code Figure 3

Text

Description automatically generated

# Works Cited

"TensorFlow Lite for Microcontrollers." TensorFlow, Google, 2021, <https://www.tensorflow.org/lite/microcontrollers>.

"Smart Pointers in C++." GeeksforGeeks, 5 Sept. 2021, <https://www.geeksforgeeks.org/smart-pointers-cpp/>.

"Top 5 C++ Security Risks and How to Avoid Them." Snyk, 22 Feb. 2021, snyk.io/blog/top-5-c-security-risks/.

"Finding and Fixing C Vulnerabilities." Secure Coding. Secure Coding Inc., n.d. Web. 01 May 2023. <https://www.securecoding.com/blog/finding-and-fixing-c-vulnerabilities/>.

Snyk. “Buffer Overflow Attacks in C: What They Are and How to Defend Against Them.” Snyk Blog, 16 June 2020, <https://snyk.io/blog/buffer-overflow-attacks-in-c/>.

"std::unique\_ptr - cppreference.com." Cppreference.com, 2021, https://en.cppreference.com/w/cpp/memory/unique\_ptr.