*Implementation of an Open-World Recognition-capable Network Intrusion Detection System*

Vision

# 1.                  Introduction

# This is the vision document for Implementation of an Open-World Recognition-capable Network Intrusion Detection System, IOWR-NIDS. IOWR-NIDS is the implementation of a network intrusion detection system using machine learning to identify internet packets with Open-World capability. Open-world capability is the ability to identify gaps in training knowledge and create temporary identifiers for new or unseen data. These temporary labels should be able to aid in the identification of the packets by trained network intrusion specialists.

# Current machine learning intrusion detection systems do not include a method of informing the specialist that the packet is not related to the training data for the model. This can lead to packet identification being unhelpful in recognizing new network attacks, possibly giving misleading information. Open World recognition should aid with identifying those attacks.

# Additionally, we will be including a method of detecting shifts in packet makeup to create a degradation detection mechanism. This mechanism will warn when the training data has become outdated and that the model needs to be retrained on the current state of internet packets.

# 2.                  Positioning

## 2.1               Problem Statement

|  |  |
| --- | --- |
| The problem of | Computer Network Attacks |
| affects | Real-world military networks |
| the impact of which is | Threats to networked computers and sensitive government data |
| a successful solution would be | Alerts based on package classifications (benign, malicious, unknown) and confidence of these classes |

## 2.2               Product Position Statement

|  |  |
| --- | --- |
| For | System administrators of the Army’s Cyber Security |
| Who | Need a tool to keep malicious packets off their network |
| The IOWR-NIDS | is a front-end for open-world packet-categorizing software |
| That | Makes it easier to catch malicious network traffic |
| Unlike | Signature-based network intrusion-detection systems |
| Our product | Is faster and more adaptable due to its open world recognition |

# 3.                  Stakeholder Descriptions

## 3.1               Stakeholder Summary

| **Name** | **Description** | **Responsibilities** |
| --- | --- | --- |
| Army Cyber Institute | Support the Army in the domain of cyberspace | Provides funding, product guidance |

**3.2**  **User Summary**

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | **Description** | **Responsibilities** | **Stakeholder** |
| Sys/ Network Admins | Knowledgeable user, the direct end user | Acceptance testing. System is developed to suit their needs | The Client represents the Network admins in the stakeholders |

## 3.3               User Environment

Users will be working in an office environment with access to high powered computational devices. System processing speed and memory are not concerns. Work will be done on a Linux machine, with no indication of being used on different systems in the future. The system should be fast to respond and able to process large amounts of network data.

The product is not intended to be deployed at enterprise scale. An estimate of 50 people will be involved in the use of the tool.

## 3.4 Summary of Key Stakeholder or User Needs

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Need** | **Priority\*** | **Concerns** | **Current Solution** | **Proposed Solutions** |
| Installation and usage manual | 3 | None | N/A | Create a usage manual for IOWR-NIDS. |
| Testing and assurance documents | 4 | None | N/A | Test IOWR-NIDS using testing dataset. |
| Performance report | 3 | Undefined metrics | N/A | Create a performance report using testing dataset. |
| Display stats of detected packets and model health using Dash | 2 | Dash integration | Suricata | Dashboard in Dash using Plotly charts for visualizations. |
| Network/model interactions through the Dash interface | 2 | Thread communication. | N/A | Poll for results from the model.  Dequeue results from the model. |
| Detecting degradation of Neural Network | 3 | Research Ongoing | N/A | Solution based on current research results. |
| Replace/retrain Neural Network when threshold is reached for replacement | 4 | Undefined meaning | N/A | No solution has been created yet. |
| Apply a language model for unknown packets | 4 | Research communication | N/A | Need to communicate with the researchers related. |
| Implement Neural Network | 2 | Needs reformatting | N/A | Reformat Neural Network from research to work continuously. |
| Packet Sniffer | 2 | Data formatting | Suricata | Wireshark |
| Wireframes | **1\*\*** | None | N/A | Draw.io  Microsoft Visio |

\* Note: Priorities **not** declared by client

\*\* Note: Priority was declared by client

## 3.5 Alternatives and Competition

Signature-based network intrusion-detection systems: large numbers of threat types significantly slow down performance. Tooling already exists and the client has access to it.

Firewall: Significantly less versatile and robust. Extremely easy to setup and cheap to run.

Suricata: A commercial, rule-based intrusion-detection system

# 4. Product Overview

The IOWR system should be able to identify classes of network attacks identified during training in addition to being able to highlight changes to the composition of network packets over time.

## 4.1 Product Perspective

The IOWR-NIDS system should be a self-contained network intrusion detection system with dependencies on python-dash, python-torch, and a packet collection system. It interacts with network intrusion specialists to identify weaknesses and attacks in a networked computer system. In accordance with that, it would be best to use this in coordination with a packet identification program.

## 4.2 Assumptions and Dependencies

We are assuming the following:

* We can collect data in a proper format to use the available labeled data to train our model.
  + Should this be incorrect we will need to label the data ourselves, which will take a lot of time.
* We can create temporary labels that are meaningful.
  + Should this be incorrect, the IOWR tool will lose some of its usefulness.
* We can use a meaningful value to assess model assurance.
  + Should this be incorrect we will default to the number of unknowns detected for model assurance.
* We assume this project will only be used by system admins.
  + Should this be incorrect we will need to create a more understandable interface due to being used by non-advanced practitioners.
* We assume this will be used on a Linux operating system.
  + Should this be incorrect we will need to acquire testing software that can simulate the correct operating system.
* We assume this will be used on high powered devices.
  + Should this be incorrect we will need a smaller machine learning model design.
* We are assuming nothing else will go wrong.
  + Should this be incorrect we will deal with problems as they appear.

# 5. Product Features

* User interface written with Python using Dash.
  + The UI would be useful for displaying the number of packets identified, unidentified, and their labels.
  + The system would alert for specified packet categorizations.
  + Visualizations (pie charts, histograms, etc.) for the packet categorizations.
  + The confidence of each result would be indicated in the visualizations.
* Collect network data and classify individual packets.
  + It is the primary basis for the entire software, enabling it to analyze packets.
* Actively monitor the network to detect attacks.
  + The tool would be much more effective if it’s constantly monitoring network traffic with no downtime.
* Measure assurance level of the current performance
  + The program needs to be accurate and consistent with its labels.
  + The program will also need to be able to assess if the current data is similar to the data the model was trained on.
* Detect performance degradation and fix the algorithm accordingly
  + Improving the program’s performance and accuracy increases its ability to act in real-time on network packets.

# 6. Other Product Requirements

The software is expected to be run on extremely high power computers in regards to CPU, GPU, and RAM. Performance in terms of time and memory complexity is not a primary concern because of the hardware resources available. That said, it must still perform very well in this environment. Real-time network scanning and analysis is a requirement, with tools to appropriately handle necessary changes to the underlying algorithms on the fly. The known constraints are that the software must perform on Linux, must be written in Python, and the GUI framework is Dash.

The software must be extremely reliable. It cannot stutter, crash, display visuals incorrectly, mislabel data, or any other similar errata. The final product must be entirely stable.

The client expects installation manuals and tooling, as well as performance, testing, and assurance reports.