ProDefense – A Binary Similarity Model for Malware Classification

Research Deliverable
Prepared by **ProDefense Capstone Team**

Reverse Engineering
Gursharan Singh, Karanpreet Singh

Data Engineering
Krystian Bista, Rishik Kolli

Machine Learning
Darci Vincent , Riley Hall

ProDefense

Agenda

3 Purpose

4 Introduction / Requirements

6 Project Scope

7 Malware / Sample Selection

8 Reverse Engineering

12 Smoke Loader and ZBot Findings

14 Feature Engineering

16 Neural Network Training

18 Value Created

19 Takeaways / Next Steps

Purpose

Enhancing our software's ability to automatically detect potential malware using machine learning algorithms is crucial to effectively combat the ever-evolving threats to our computers, networks, and data security, reducing the workload and speeding up the detection process.

According to IBM's 2022 Data Breach Report, 83% of organizations experiences more than one data branch during 2022

According to Verizon's 2022 Data Breach Investigations Report, the total number number of ransomware attacks surged by 13%

Binary Similarity Model

- A Binary Similarity Model is an analytical approach of examining and computing the similarity of some input (i.e. functions, files) against some standard of comparison.
    - This project is a perfect example insofar as some unknown program file can be examined and its similarity to **known** malware can be computed; if such similarity is high, then it is likely that such a file is malware and of that malware type.

- With a plethora of malware variants and benign programs, it is imperative to train a neural network that will compute the degree of similarity between some given input against malware inputs that the neural network was trained on to detect.

Requirements

- Fundamentally, the key necessities of the output of this project is a functional neural network(s), that can read in files consisting of program data, and compute a similarity score of the given file against some malware files of various types.

- This similarity score would fundamentally **place** the input file within different malware families, or not at all if it is not malware.

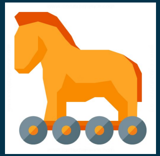
- The remaining core topics of this project, such as technology stacks and tools, are up to research and design decisions made by the team throughout this course.

Project Key Deliverables

- At its core, this project requires three fundamental phases:
- First – Malware Selection, Reverse Engineering, and Data Gathering
- Second – Feature Engineering, Data Formatting & Preparation for the Neural Network
- Third – Development, training, and testing of the Neural Network

Malware Selection

- Goal – Finding the most popular and widespread type of malwares.
- Findings – Remote Access Trojans where one of the most popular forms of Malware types within all the different types but also within the Trojan Family.
- Researching for particular malwares, Smoke Loader and Zbot where found on many different lists from government websites, to hacker forums.

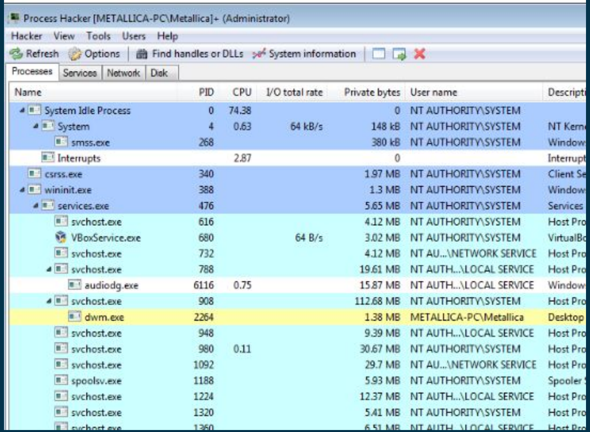


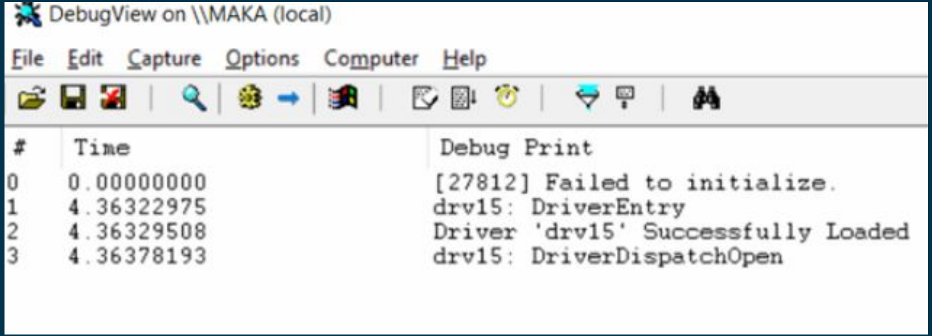
Reverse Engineering

- Detect It Easy – Portable Executable (PE) packers detection tool, which allows us to analyze the malicious and suspicious content of malware binaries.
- IDA – Interactive disassembler allows us to reverse engineer and analyze executable files such as malware binaries, also offers control glow graph view, and scripting support.
- Sandbox Environment – Windows 10 machine for analyzing malware, in a safe environment.

Additional Tools For Reversing/Research

- WinDbg: Kernel Debugger
- Process Hacker 2
- Various IDA Python Plugins to aid reversing
- ChatGPT, for translating asm to readable c++ code





Data Gathering

- Goal – Accrue as many samples/strain of each family of malware family for both Zbot and Smoke Loader.
- Findings – Acquired sample strains for malware analysis.
    - VirusTotal
    - Malware Bazar
    - VX Underground

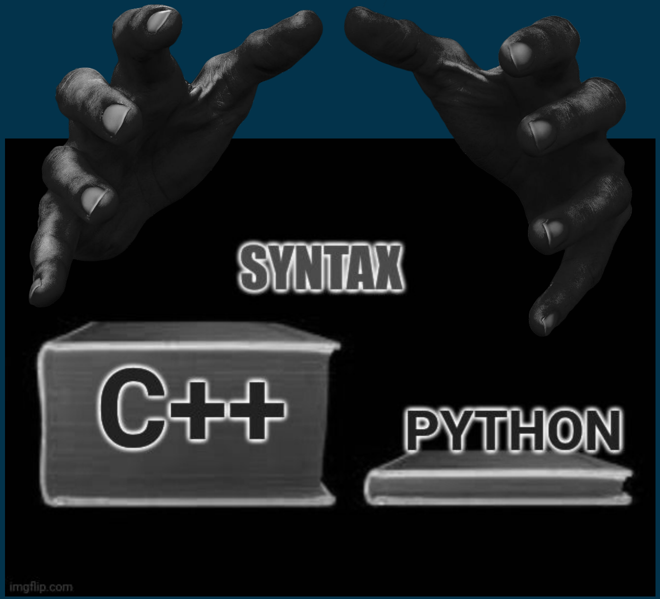
Programming Language

- C++:

  – Windows API, able to read/dump file memory

  – Most malware are written in C/C++

  – Robust, low level language. Best of memory reading/write/ exploiting.



Windows API can be used by anyone with line of code to read another programs memory.

Smoke Loader: The first findings

Smoke Loader: Analysis thus far

```
.rdata:00429C18          db 2Ch          ; shellcode start
.rdata:00429C19          db  0Ah
.rdata:00429C1A          db 31h
.rdata:00429C1B          db  33h ; 3
.rdata:00429C1C          db 8
.rdata:00429C1D          db  14h
.rdata:00429C1E          db  16h
.rdata:00429C1F          db 13h
.rdata:00429C20          db  1Fh
.rdata:00429C21          db 1Eh
.rdata:00429C22          db 3Dh
.rdata:00429C23          db  33h ; 3
.rdata:00429C24          db  3Dh ; =
.rdata:00429C25          db  0Fh
.rdata:00429C26          db    2
.rdata:00429C27          db 2Bh
.rdata:00429C28          db  29h ; )
.rdata:00429C29          db 21h
.rdata:00429C2A          db  2Bh ; +
.rdata:00429C2B          db 35h
```

```
push    eax                 ; OLD_PROTECTION
push    [ebp+flNewProtect] ; PAGE_EXECUTE_READWRITE
mov     dword_4615EA, 74636574h
push    dwSize              ; dwSize
mov     dword_4615E6, 6F72506Ch
push    dword_45CF08        ; SHELLCODE ADDRESS
mov     word_4615E0, 6956h
mov     byte_4615EE, bl
call    ds:VirtualProtect
```

```
ntdll.dll
    NtUnmapViewOfSection
    NtWriteVirtualMemory
kernel32.dll
    CloseHandle
    CreateFileA
    CreateProcessA
    ExitProcess
    GetCommandLineA
```

```
push    ebp
mov     ebp, esp
sub     esp, 8
push    ebx
push    esi
push    edi
push    0D5786h         ; LoadLibraryA
push    0D4E88h         ; kernel32.dll
call    find_function
mov     [ebp+var_8], eax
push    348BFAh         ; GetProcAddress
push    0D4E88h         ; kernel32.dll
call    find_function
mov     [ebp+var_4], eax
jmp     loc_1F742
sub_1F65B endp
```

Feature Engineering, Data Formatting, and Preparation

- Examining common and comparing features pertaining to non-malicious programs and those that are malicious.
  - Static analysis of two malware families to start with: **SmokeLoader** and **ZBot**.

- Choosing feature categories to develop a **feature model**.
  - Developed a simple, but wholesome set of features for initial extraction and experimentation.

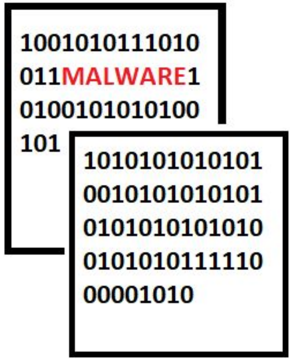
- Developing **Python** scripts to automate development of neural network input files built using **NumpPy** arrays and **Pandas** dataframes.

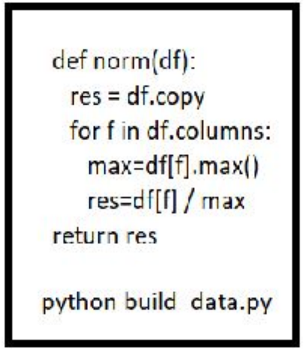
Feature Engineering Process Diagram

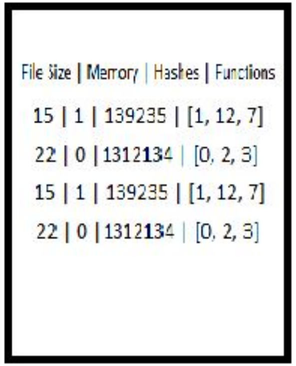
Reverse Engineering and Data Extraction

Feature Extraction and Formatting

Feature Matrices = Neural Network Input



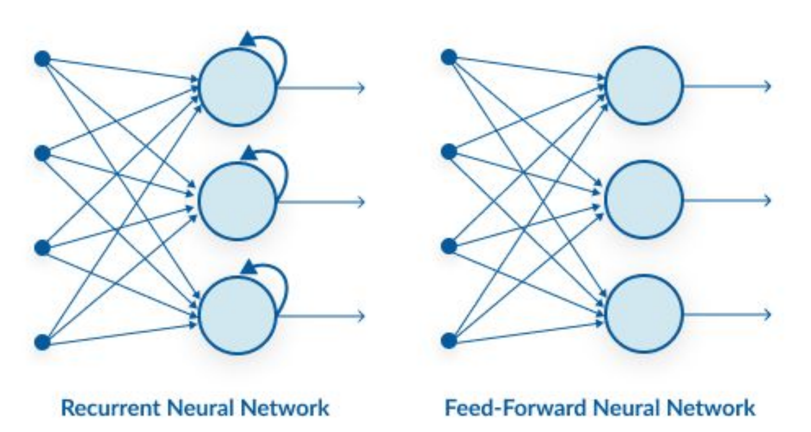




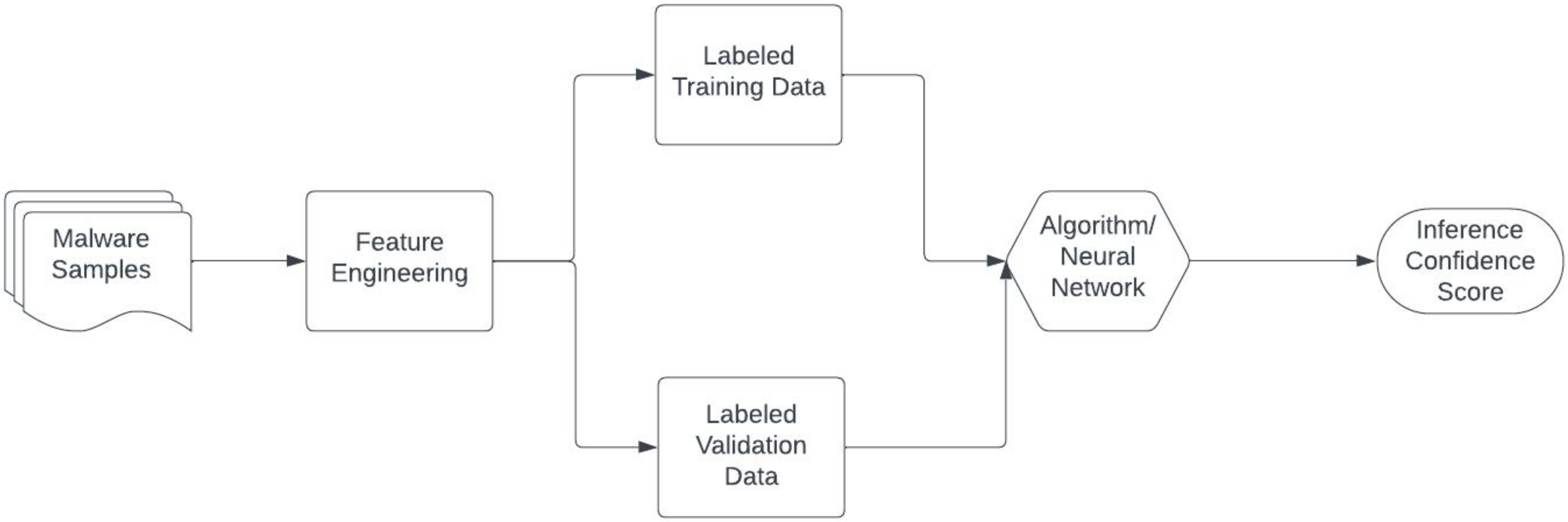
Development, Training, and Testing of the Neural Networks

Looking into...

- Pytorch
- VulSeeker
- Gemini
- BinFinder
- Algorithms:
  - Random Forest
  - KNN
  - XgBoost
    - We will be comparing multiple algorithms against one another to determine which is most accurate



Machine Learning Diagram



Value Created

- Automates the detection process of malware files, indicating whether a file is malware or not.
    - Enables the customer to be free from having to manually perform static analysis on program files.

- Categorizes the type of malware family that a given file may belong to in addition to indicating whether a file may be malicious or not.
    - Provides some level of insight into the type of malware provided, thus giving the customer a better understanding of the malware input at hand.

Lessons Learned

- The complexity of breaking down a project into requirements, organizing team functions, determining tasks per sprint.
- The heavy assortment and needed refinement of features that can be used to classify a binary as either malware, belonging to a specific malware family, and being non-malicious.
- Different types of neural networks, their purposes, strengths and weaknesses.
- Further Optimization of ML model requires considerable data.
- With number of neural network types that are available, choosing the one to fit the data problem proved to be challenging.

Future Plans for Next Semester

- To determine the robustness of the current prototype.
  - Begin experimenting with hyperparameters such as the depth of the neural network, activation function, and number of inputs.

- Coupled with researching other open source machine learning models that we can modify using our data inputs to train.

- Reconsideration, evaluation, and enhancements of the feature model.

References

https://hbr.org/2023/05/the-devastating-business-impacts-of-a-cyber-breach

https://www.cisa.gov/news-events/cybersecurity-advisories/aa22-216a

https://www.crowdstrike.com/cybersecurity-101/malware/types-of-malware/

https://www.ibm.com/topics/knn#:~:text=The%20k%2Dnearest%20neighbors%20algorithm%2C%20also%20known%20as%20KNN%20or,of%20an%20individual%20data%20point