MLCS – fall 2023
Assignment 1

Instructor: Hans P. Reiser

1 General remarks

The final project aims to delve deeper into specific topics covered during the lecture. You will have the opportunity to work collaboratively in teams of two. A common focus of all projects will be on a practical implementation task. You are expected to submit a final report in the style of a research paper and present your key results in a short presentation/demo. We are planning for these presentations to take place on Nov 30th and/or Dec 1st (in case of conflicts with a 3-week course, arrangements for a later date can be made).

Code Submission Rules:

- Make sure your results are reproducible, i.e. include all essential parts.
- Do not include complete datasets provided to you, nor other irrelevant parts.
- Clearly indicate which dataset(s) you have used in your project.

Report Submission Rules:

- Create a self-contained report that tells your project's story. This should include an introduction briefly explaining the problem your project addresses; a description of the approach you've taken to tackle the problem; and a presentation and discussion of the results you've obtained.
- Your report should follow the style of a research article.
- Format your report using the two-column ACM SIGCONF style (Word and LaTeX templates available at `https://www.acm.org/publications/proceedings-template` and also directly accessible on Overleaf – use `\documentclass[sigconf]{acmart}`)
- While there are no strict length restrictions, about 6-8 pages should be reasonable. If in doubt, seek feedback on a draft version 1-2 weeks before the final deadline.
- Document how you collaborated within your team (in case of team projects).

Note that the expected workload for this project is around 48 hours. While perfection may not be feasible in this time frame, the project's goal is to make a reasonable effort in addressing the challenges presented. You are not expected to spend significantly more time than that, but your code and in particular your report should reflect that you made a decent effort in tackling the project effectively.

We also plan to provide some lab assistance class to help you get started / avoid being stuck in minor issues at the beginning.

2 Projects

2.1 Malware classification on API traces (x2)

In this project, you will work on a unique dataset comprising approximately 50,000 recent traces of Windows malware. Each sample is annotated with the hash of the executable file. This project can be undertaken by two separate groups:

Group 1: Your objective is to utilize the VirusTotal API (using the file hash) to obtain the "popular threat classification" labels for a subset of the dataset (note that the free API allows 500 requests per day per person, making it impractical to classify all samples. Instead, focus on a subset containing a minimum of 1000, but preferably 2000 samples).

Group 2: Your task involves using annotations that are provided, which include fine-grained results of Yara rules. In these annotations – these are JSON files mapping tags to arrays of hashs – each sample may have multiple tags, with a total of 187 possible tags. (You may later decide to group multiple tags into labels, or ignore some tags).

The tasks for both groups are as follows:

- Data Preprocessing: The dataset is structured with one file per sample, containing one line per API call. Extract the API call names from the traces to prepare the data for analysis. (You may collaborate with the second group on this task to preprocess the dataset.)
- Data Splitting: Divide the dataset into training, validation, and test subsets. Use the validation subset to find the "best" approach, and use the test subset only for the final evaluation.
- Machine Learning Classifier: Your primary goal is to develop and train a machine learning classifier that can predict labels for API call traces based on the data and annotations available.

In A3, it turned out that the best solutions generally used a simple bag-of-words (or rather bag-of-API-calls) approach with a random forest classifier. An excellent project is expected to make another attempt to explore appraoches that consider the order of / sequences of API calls as a basis to improve classification.

2.2 Malware classification with the Cuckoo sandbox (x2)

Disclaimer: In this project, you will work with real malware samples. For the sake of security, both yours and that of others, you acknowledge and agree to take the utmost care to ensure that no malware samples are executed outside the designated sandbox. You also understand the importance of not misusing any insights gained during this project to harm or compromise any individuals or systems.

Disclaimer 2: A challenge in this project arises from the fact that Cuckoo 2.x is no longer maintained and relies on Python 2.x. It primarily targets Windows 7 as the malware execution environment. Cuckoo 3.0 is currently under development (available on a Git repository), but no stable release is available as of now. You should be prepared to invest significant effort into getting the setup (with either one of the two) to work. Additionally, you will need access to have either a bare-metal Linux system on which you can run Windows 7 VMs, or you will need to deal with the joys and complexities of nested virtualization (running a Windows VM within a Linux VM).

In this project, your primary objective is to install and utilize the Cuckoo sandbox to collect information about malware samples. You have the flexibility to choose where to obtain these malware samples (the website `https://github.com/topics/malware-samples` is a recommended starting point). Furthermore, two groups working on this project can collaborate on setting up Cuckoo and obtaining and running malware samples.

Common tasks for both groups: Utilize the VirusTotal API with the hash of the executable malware to obtain "popular threat classification" labels. Please note that the free API allows for 500 requests per day per person, limiting the number of malware samples you can analyze.

Group 1: API call traces classification: Your task is to train a ML model to classify malware samples based on API call traces collected by the Cuckoo sandbox.

Group 2: Classification based on other information: Your task is to train a ML model based on any other available information collected by Cuckoo (e.g., static analysis results, network traces).

This project presents a unique opportunity to engage with real malware samples, although the challenge of working with older software and complex virtualization setups should not be underestimated.

2.3 Using Machine Learning in Main Memory Forensics

Precise fingerprinting of an operating system (OS) plays an important role in applications like penetration testing, intrusion detection, and memory forensics. For analysing a main memory dump of a system, you need to have knowledge about the data structure layout used by the system, and OS kernel data structures are a particularly valuable source to extract information about running processes, open files, and network connections.

The paper "OS-Sommelier" (`https://dl.acm.org/doi/10.1145/2391229.2391234`) by Gu et al. (SoCC '12) proposed a method to precisely fingerprint cloud guest OSes based on physical memory dumps. However, the landscape has evolved since then, with newer 64-bit systems and developments such as Kernel Address Space Layout Randomization (KASLR) presenting fresh challenges.

Project tasks:

- The first objective is to reproduce the results of the OS-Sommelier paper on more modern systems. This requires updating the approach to accommodate 64-bit systems and addressing any potential challenges. The original OS-Sommelier code is open source (`https://github.com/utds3lab/OS-Sommelier`). Also, a prototype implementation (from a UROP project at RU) for 64-bit systems is available.
- Experiment with enhancing that approach using machine learning classifiers: Investigate the incorporation of machine learning classifiers to improve the accuracy and efficiency of OS fingerprinting (ideally, even in the presence of a malicious subject manipulating the target system).

A large, labelled memory dump dataset is available for testing and evaluation.

2.4 Using Machine Learning in Main Memory Forensics, Part II

As previously mentioned, the ability to identify OS kernel data structures within a memory dump is important for various cybersecurity and forensics applications. The SmartVMI research project explores ways how to achieve this without prior knowledge of the specific OS on the target system. Instead, it explores machine learning approaches to detect patterns within the main memory, offering a potential path for the automatic identification of kernel data structures.

In this project, your task is to explore reconstructing information about a target system based on a raw memory dump. A starting point is the extraction of features by tracing and following pointers within the memory dump. In memory, pointers form a complex graph structure, representing a hierarchy of data relationships.

On this basis:

- You can explore the classification of memory by leveraging machine learning classifiers on features extracted from the pointer graph.
- In addition to pointer graphs, you can explore the potential of using other memory features for classification, such as statistical properties.

This project presents a challenging task. The goal here is not to find the perfect solution, but to experiment with different methods to explore their effectiveness empirically. You should have some background on operating systems internals, data structures, and pointers.

2.5 Static analysis of memory dumps of a running system

In our lecture, we've explored both static and dynamic approaches to malware detection and classification. This project looks into a hybrid approach combining these two paradigms. It involves running a malware sample dynamically and capturing a memory snapshot, from which static features are then extracted for analysis.

This combination tries to combine the benefits of dynamic and static analysis. When a malware sample is executed dynamically, it will unpack, decrypt, and undo obfuscation techniques commonly employed during the packing of malware into an executable. Due to this, the memory snapshot will give better access to some features hard to obtain from the executable file. Static analysis of the main memory snapshot then can leverage methods similar to those discussed in our lectures for static analysis.

This approach has been described in the paper "Detecting Obfuscated Malware using Memory Feature Engineering" by Carrier, Tristan, Princy Victor, Ali Tekeoglu, and Arash Habibi Lashkari (Int. Confe on Information Systems Security and Privacy, 2022). The dataset used in that research is available at `https://www.unb.ca/cic/datasets/malmem-2022.html`

The tasks of this project are as follows:

- Using the public data set, explore to what extent you can reproduce, in a ML-based classifier implemented in Python, the results of the paper.
- Can you validate the results with data (memory snapshots) from some other source? For example, you can have a look at data from `https://www.ncbi.nlm.nih.gov/pmc/articles/PMC6743008/`

2.6 Intrusion Detection for IoT Infrastructures

In the area of Internet of Things (IoT), security is an important concern. This project makes use of a novel IoT attack dataset (CIC IoT Datset 2023) designed to advance security analytics applications in real IoT operations.

The dataset is available at `https://www.unb.ca/cic/datasets/iotdataset-2023.html` and involves executing 33 different attacks within an IoT topology comprising 105 devices.

Project goal: Your goal is to use machine learning approaches to classify this IoT attack dataset with a high degree of accuracy. Specifically, you have the following objectives:

- Data preprocessing: Select and implement suitable data preprocessing techniques to prepare the dataset for analysis.
- Algorithm selection: Explore and experiment with a variety of machine learning algorithms to determine which one works best.
- The main goal is to find an approach that yields the best average F1 score for the classes ”Spoofing”, “Brute Force”, and “Web-based”.
- Performance optimization: Fine-tune the selected algorithm(s) to maximize their classification performance.
- Final evaluation with the test set.

2.7 Federated Learning for NIDS

In assignment 2, we used a centralized approach to NIDS (one central nodes has all data available, trains a model, and classifies traffic). In practice, for privacy as well as for performance reasons, a central collection of data is not feasible or not desirable.

Explore, based on a literature search, ways to implemented decentralizes learning / federated appraoches for network intrusion detection. Preferrably, use the same data set as used in Assignment 2.

- Select some approach for decentralized/federated learning and explain how it works
- Implement and train an IDS based on federated learning
- Compare the performance to a centralized approach

2.8 Browser Plugin for Solving Captchas

[reserved]

2.9 Your own ideas

[feel free to propose own ideas]