Malware Analysis Project

Spring 2023

Introduction:

The goals of this project:

- Familiarize you with the types of behaviors that can be displayed by real-world malware samples and how to safely analyze these behaviors using JoeSandboxCloud (https://www.joesandbox.com/).
    - Joe Sandbox detects and analyzes potential malicious files and URLs on Windows, Android, Mac OS, Linux, and iOS for suspicious activities. It performs deep malware analysis and generates comprehensive and detailed analysis reports.
- Familiarize you with machine learning concepts applied to malware analysis.
    - Build on your malware analysis experience with machine learning strategies.
    - Get familiar with malware behavior clustering, and how to use it to classify malware.
    - Build a machine learning based malware classification tool using Malheur. For more details about Malheur you can visit Malheur manual and read more about it.

Additional information:

- Phase 1 will be submitted in Gradescope.
- Phase 2 will be submitted in Gradescope.
- Make sure you have a computer with sufficient horsepower to run the Project's VM. Minimum RAM required for this project is 4 GB RAM for the VM, 8GB RAM on your host.
- We offer a Frequently Asked Questions (F.A.Q.) thread in Ed Discussions here.
- The F.A.Q thread will be constantly updated. Therefore, before asking anything, take a look at it, and, if your question is not covered already, feel free to post it as a reply in the same thread.

Accessing project resources:

Setup (0 points)

1. Download the project VM here:
   https://cs6035.s3.amazonaws.com/cs6035_p2_p3_p4_spring2023.ova

   Please note that the file is over 13GB, so it will take some time to download. Do not wait until the last minute to download it. Do it right away!

   SHA-256 for this file:
   6C1119C82B18B5C2CB87D52D708FAA9CFDE80D319E3C44C2D9A36C9D603AA859

2. Students need a x86 (Intel) machine to properly run the project virtual machine (please see the Ed Discussion post regarding VM Troubleshooting new link to Ed if you have any issues regarding your virtual machine).

3. Import the OVA into VirtualBox. Note that this may be easiest to do by double-clicking the OVA file and letting the file association open VirtualBox.

4. Log in to the newly imported VM using these credentials:
   - Username: debian
   - Password: debian

5. Run the following commands in the Terminal on the VM:

   - ```
     wget --no-check-certificate
     'https://docs.google.com/uc?export=download&id=1DAoRPKQ6DxDdC2EP
     ZSkr0A1wgQ0hj1Y9' -O Desktop/malware_analysis_reports.zip
     ```

   - ```
     unzip Desktop/malware_analysis_reports.zip -d
     Desktop/malware_reports
     ```

6. The five malware files will be located in the “malware_reports” directory on the Desktop.

Phase 1 (50 points):

Analyze your malware samples (50 points)

You will investigate and label some of the more sophisticated malware behaviors from the five malware samples we provided. Use the included JoeSandbox reports to identify the malware's behavior. Note that malware samples can share behaviors. So initially you should assume that each malware we question you about below has every behavior listed. It's your job to determine if that assumption is actually true.

Hint: Look at the API/system call sequence under each process generated by the malware sample and determine what the malware is doing. Note that each JoeSandbox report may contain multiple processes with many different system call sequences. If any of the behaviors are seen (or attempted, but not necessarily successful) in any process in the report, then that malware has attempted that behavior. This is, of course, not completely practical, as legitimate applications may perform the same actions in a benign fashion. We are not concerned with differentiating the two in this assignment, but it is some food for thought.

Clarification for attempted: We mean by “attempted” that a specific action was attempted but failed. By “specific” we mean that it is clear which action is attempted. If you have a registry key, for instance, that is unambiguous (like, say, it is used only to set a startup option), but it fails to change the key, that is an attempt for our purposes. But if you have a more generic registry key that governs multiple settings, we don’t know for sure which key or keys it is attacking and so the action would not count as an “attempt”.

You will encounter that the same API functions can end with either a W or an A. This is a standard practice in the Windows API, and this document explains the difference (either one could in theory be present in the wild):
https://docs.microsoft.com/en-us/windows/desktop/intl/unicode-in-the-windows-api

For each of the following questions, mark which of the malware exhibit the identified behavior:

1. Attempts to get victim to disable security protections
2. Microsoft Office key deletion
3. Microsoft Excel key creation
4. Creates registry values (any)
5. Drops RegAsm virus
6. Issues signal to cause immediate program termination
7. Malicious file most likely programmed in C or C++
8. Detects the Mirai botnet
9. Keylogger attempt

10. Attempts to copy clipboard

11. Hooks registry keys/values to protect autostart

12. Possible PFW / HIPS evasion

13. Uses the Windows core system file splwow64.exe

14. Drops a portable executable file into C:\Windows
    a. The term "drop" in the behavior "Drops file(s)" means to create (or attempt to create) files, not to delete files.
    b. We are just looking for dropped files for this behavior.

15. Looks for the name or serial number of a device

16. Attempts to obscure the meaning of data as an added layer of data

17. HTTP GET or POST without a user agent

18. Uses loops or otherwise needless repetitions of commands, such as Pings, used to delay malware execution and potentially exceed time thresholds of automated analysis environments.

19. Attempts to override the domain name system (DNS) for a domain on a specific machine.

20. Possible system shutdown

DELIVERABLE: Your deliverable for this part of the assignment will be your final JSON file with your answers to the 20 questions.

Download the submission template or use the JSON format below for your answers:

```
{
  "behavior01": "",
  "behavior02": "",
  "behavior03": "",
  "behavior04": "",
  "behavior05": "",
  "behavior06": "",
  "behavior07": "",
  "behavior08": "",
  "behavior09": "",
  "behavior10": "",
  "behavior11": "",
  "behavior12": "",
  "behavior13": "",
  "behavior14": "",
  "behavior15": "",
  "behavior16": "",
  "behavior17": "",
  "behavior18": "",
  "behavior19": "",
  "behavior20": ""
}
```

The submitted answers should be in the format (**this is an example only**):

```
{
  "behavior01": "2,4",
  "behavior02": "3",
  "behavior03": "1,2,3,4,5",
  "behavior04": "4",
        .
        .
        .
}
```

The naming of the submission file is not important, as long as it is JSON ("submission.json" is an example). You will have 20 attempts to submit your answers. If you attempt to make more submissions than the limit, your grade will be a ZERO for this Phase. You will be able to choose your best submission of the 20 manually in Gradescope, but this **MUST** be done **BEFORE** the project deadline. No late submissions or requests to update the submission will be accepted after the project deadline. Please submit the answers in the JSON file in the Gradescope assignment **Project Malware Analysis - Phase I**.

Phase 2 (50 points):

Background

In this phase you will learn how to apply Machine Learning concepts to malware classification. You'll be given a dataset of malware samples. Using Malheur, the software used for clustering malware in this project, you'll run an unsupervised learning clustering algorithm in order to classify them by behavior.

Understanding malheur (0 points)

1. Malheur uses a hierarchical clustering method to cluster malware samples. However, due to the number of features the malware has (there are many dimensions to the malware data), just performing a hierarchical clustering will take too long. In order to complete this assignment, and become accustomed to how Malheur works, you should read the following sources about malheur:
    - Malheur original paper
    - Malheur manual

2. Pay special attention to malheur configuration parameters. You are given a malheur sample configuration file to serve as a starting point. You will be asked to work with these parameters (not necessarily all of them) during this assignment. Nevertheless, some of these parameters should remain unchanged, since changing them will be detrimental to the assignment's objectives. This way, unless you know exactly what you are doing, you should not change values for:

Parameter	Value
`generic.input_format`	`"text"`
`generic.event_delim`	`";"`
`generic.state_dir`	`"./malheur_state"`
`classify.max_dist`	`1.00`

All other values can be changed in the configuration file. Refer the malheur manual for specifics on each configuration parameter.

A special note about ngram_len

Each semester, many students are concerned about the value of ngram_len in the configuration file and how it relates to this project. The ngram_len parameter is one of the parameters that can be changed in the configuration file, and you may submit any value for this parameter. The malheur manual states the following about ngram_len:

"This parameter specifies the length of n-grams. If the events in the reports are not sequential, this parameter should be set to 1. In all other cases, it determines the length of event sequences to be mapped to the vector space, so called n-grams."
Malheur manual

While the malware behavior is encoded by listing all API calls in sequential order, (see **Understanding the dataset** below) if you receive better performance by selecting a different parameter value than the default provided to you, you may select another value.

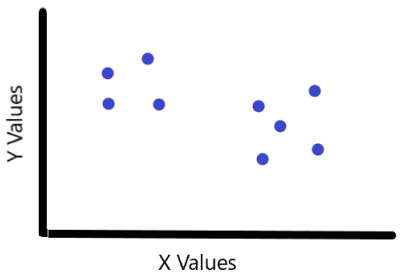
Understanding Unsupervised Learning and Clustering (0 points)

1. Unsupervised Learning is a type of Machine Learning used to extract information from a dataset. It is considered unsupervised, because no prior information on the structure of the dataset is provided to the algorithm. Instead, the unsupervised learning algorithm attempts to extract this information. One of the most common types of unsupervised learning is called clustering. Clustering looks for clusters of data-points in the dataset, and assigns those to their own group (called a cluster).

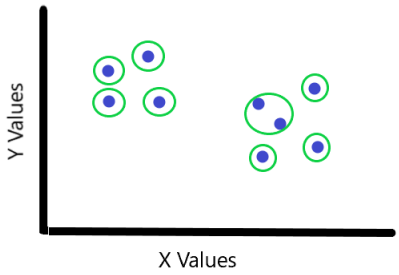
2. There are many kinds of clustering used in data analysis. Malheur, the software used for clustering malware in this project, uses a hierarchical clustering technique. Hierarchical clustering begins by setting each entry in the dataset to be its own cluster. Then, it combines the closest two clusters into one new cluster. It repeats this until either a target number of clusters is reached, or the distance between each cluster exceeds a target amount.

3. To further help you understand hierarchical clustering, we provide an example below. This example uses 2-d data to help you visualize what clustering is actually doing. The process of clustering malware samples is similar, but the data has many more features.

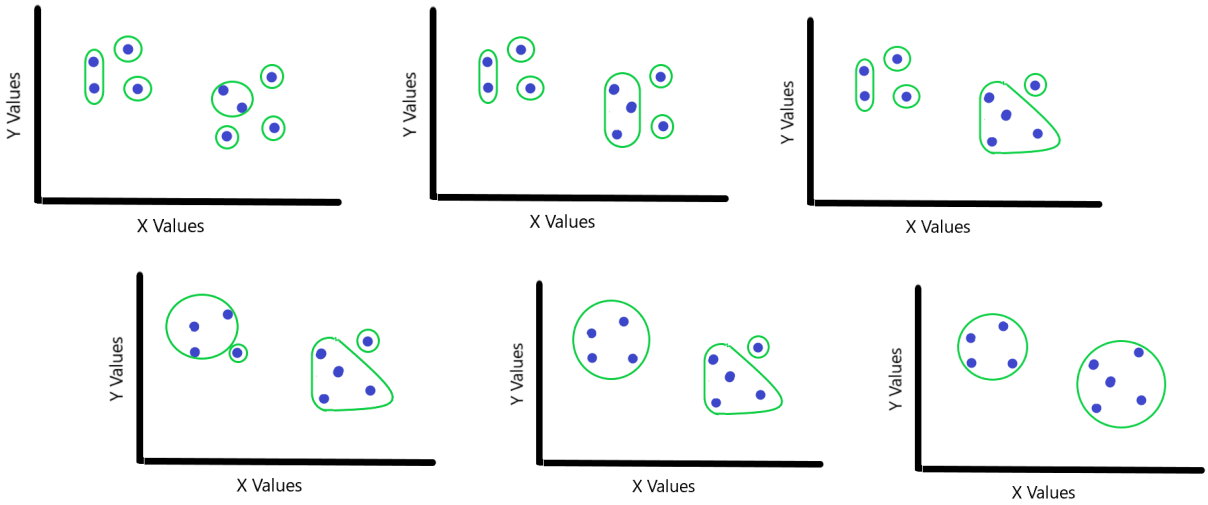
4. Let's say we're given the following data-set. It consists of x-y coordinate data that, when plotted, looks like:



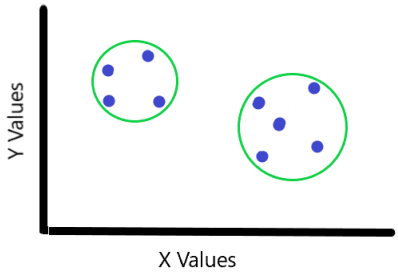
Then, we combine the closest two clusters.



And repeat the process….



We do this until each of the closest two clusters are fairly far away from one another. The exact minimum distance between the final clusters, can usually be set as a parameter to the clustering algorithm. This gives us the clusters we were expecting:



5. This is a general example of hierarchical clustering. In order to cluster malware, malheur extends this method, as you will see in the next section. In order to complete this assignment, you will have to develop a deeper understanding of what Malheur is and how it performs this clustering. We encourage you to further research Machine Learning and clustering methods, beyond this section, in order to better understand the next section.

Understanding the dataset (0 points)

1. In machine learning, a data set is a collection of data points for one of more features that represent the object of study (malware behavior, in our case).

2. Our dataset is based on information extracted from Cuckoo malware behavior reports.

3. Cuckoo malware behavior reports have a section called 'behavior'. In this section of the report, you can find all API Calls that were part of the malware behavior during Cuckoo's analysis.

4. The way we chose to encode malware behavior was by simply listing all API Calls, in the order they happened, as indicated by the report.

5. More specifically, we build a list of API call names by extracting the contents of Cuckoo JSON report fields such as 'behavior->process->[X]->calls->[Y]->api', where X is the process index number and Y is the process API call index number.

6. In the end, every feature file will be a semicolon separated list of strings, where each string is an API call name, and those API call names are displayed in the order they have actually been called by the malware during Cuckoo analysis.

7. Each sample in our dataset is labeled as to indicate which malware family it belongs to. To gather labels and assign them to our samples, we have used a malware labeling tool called AVClass (https://github.com/malicialab/avclass) along with data from Virus Total website (https://www.virustotal.com/). Because of the way malheur recognizes labels, we have placed them in each dataset sample's file name as the file extension, while the main part of the file name is the SHA256 hash code for the original malware binary. For example, take the feature file named as:

*0bc19b9304d5c409b9f480a9121c8c8abcef2f3a595ed6b2758daeb2d679b74a.**dinwod***

(Since the file extension for this malware sample is 'dinwod', it belongs to malware family Dinwod.)

8. You can use the hash code that represents the file name to research extra information about each individual dataset sample. Several Malware research related websites provide complete malware reports indexed by Hash codes. Since we have used data from Virus Total for our labels, it's probably a good place to start. You can use the URL https://www.virustotal.com/gui/file/<HASH_CODE> to directly open a malware detection report in Virus Total for the hash code <HASH_CODE>.

    - For example, to gather extra information about a dataset sample with the following file name:

0bc19b9304d5c409b9f480a9121c8c8abcef2f3a595ed6b2758daeb2d679b74a.*dinwod*

    - All you have to do is to remove the label (dinwood) from the file name and use the hash to craft and visit the following URL:

      https://www.virustotal.com/gui/file/0bc19b9304d5c409b9f480a9121c8c8abcef2f3a595ed6b2758daeb2d679b74a

9. All dataset files are located in the "dataset" folder and they are distributed in two groups: "training" and "testing".

    - The "training" folder contains files with features extracted from various malware samples based on the strategy we discussed previously. You will use those files to "train" your machine learning model using clustering techniques.
    - The "testing" folder is much smaller and contains the files you will use to test the model you will eventually create.
    - We will discuss the training and testing phases of your assignment in more details ahead.

Setup (0 points)

We have already provided the malheur binaries, datasets, and example configuration files. Once you are inside Project 2 VM, just enter the avml directory on the VM desktop using a terminal:

```
$ cd /home/debian/Desktop/avml/
```

Using malheur (0 points)

Now, it's time to get familiar with ***malheur***. As an initial warm up exercise, let's learn same sample commands:

Cluster samples (training phase):

```
$ malheur -c config.mlw -o training.txt -vv cluster dataset/training/; head training.txt
```

1. This command tells malheur to analyze the dataset contained in dataset/training/ (11,000+ samples) in order to cluster all its samples into groups of similar malwares. Since malheur is using the dataset to learn how to group malware families, we are using a special subset that is called training dataset.

2. In this phase, malheur will output quality assessment information. More specifically, it will show you three main measurements: precision, recall, and f-score. Take this opportunity to research a little bit about them. Those measurements are all related to how good our clustering has been. The way malheur does it is by looking at each sample's label. If you take a look at the contents of directory 'dataset/training/', you will see lots of files, each of which with a different file extension. Those files extensions are the labels for each sample, and malheur extracts them to see if each cluster actually groups similar samples (similar labels) and how many samples that should have been part of the group were left out. This analysis is eventually represented in numbers by precision, recall, and f-score.

Verify clustering (testing phase):

```
$ malheur -c config.mlw -o testing.txt -vv classify dataset/testing/; head testing.txt
```

3. This command tells malheur to use the clusters it has built in the training phase (using samples in dataset/training/) to classify all samples inside dataset/testing. Our objective is to decide whether the model generated in the training phase is good or bad. That is one of the reasons all samples in the testing dataset used to be part of the training dataset but were randomly selected and put apart. They are selected this way, so testing data represents reality while minimizing bias towards the training data. Think about It as it were a student preparing for an exam: the end goal is succeeding in life, but the exam will give us a rough idea if that will actually happen. The same way, malheur "studies" (analysis) and "learns" (builds a model) from the training dataset (training phase) so it can succeed in real life (classify malware samples in the wild). Nevertheless, malheur will have to pass the exam (testing phase) to make sure it's ready. Would you put the exact same questions

from study material in the exam? I hope not! Exam questions should be unique and, at the same time, representative of the real-life skills students are supposed to learn. Welcome to the testing phase!

4. In this phase, malheur will also output precision, recall, and f-score. You should think of those measurements as malheur's final grade. During this phase, malheur isn't being tested against the same exact data used to train it. It now uses a different dataset where all testing instances were statically chosen to adequately represent all training data, even though it is not the same data used in the training phase.

Cluster and Classify (50 points)

1. Work with malheur parameters in its configuration file so you can build a model to classify malware samples with at least 70% f-score during the testing phase. For that, you should run thorough, successive runs of the training phase command followed by the testing phase command. Before every run, you should adjust malheur's configuration file parameters the way you see fit, based on the knowledge you have acquired during assignment preparation (if you have skipped it, it is probably a good idea to take a step back now and read it). You can choose to do it blindly or to make judgment calls before each run. You just have to remember:
    - You have to achieve at least 70% f-score in the testing phase
2. After you have reached the 70% f-score goal, use the same model you have trained malheur with to classify each of your Project 2 original malware samples. For that, we have provided you feature files for all of Project 2 malware samples containing features extracted from their Cuckoo's JSON. Those feature files are located inside the directory "subjects" in the main assignment path (/home/debian/Desktop/avml/subjects).
3. In order to use your newly created malheur model to classify the given malware samples, make sure you are inside the assignment main directory, and run the following command:

Classify Project 2 samples (classification phase):

```
$ malheur -c config.mlw -o classify.txt -vv classify subjects/; head classify.txt
```

    - Your goal is to have all Project 2 samples properly classified (none should be labeled 'rejected') using your model with classify.max_dist set to 1. (as mentioned above, this is required)
    - If you can't achieve those classification results for Project 2 malware samples, try revisiting your training and testing phases with different parameters until you can comply with everything.
    - Also, you can take a look at Virus Total reports for the prototypes that malheur has assigned to your subjects. Seeing the full malware behavior might give you a better idea about the direction you should take.
4. By achieving all Task's goals, you will have built a tool that is able to classify unknown executable binary samples (no prior knowledge needed) into potential malware families. It is your very own AV software!
5. TASK GOALS: You have two main goals for this task:

- GOAL 1: the first goal is to achieve 70% f-score during the testing phase only (you should NOT consider f-score results for other phases for the first goal, just the testing phase).

- GOAL 2: the second goal is to classify all Project 2 malware samples with maximum distance of 1 (none should be labeled 'rejected') as indicated above.

Goal 1 is a prerequisite for Goal 2. Therefore, any results for Goal 2 are only considered valid if your configuration parameters also allow for Goal 1 to be achieved.

6. PARTIAL CREDIT: Partial credit of **25 points** will be given in case you deliver Goal 1 without Goal 2. Since Goal 1 is a prerequisite for Goal 2, you will be given 0 credits for the entire task in case you do not achieve Goal 1, no matter what you do with regards to Goal 2.

7. DELIVERABLE: Your deliverable for this part of the assignment will be your final malheur configuration file (config.mlw), the one you have used to achieve all goals (70% f-score during testing phase and Project 2 malware samples classification with maximum distance of 1). **The file needs to be labeled exactly "config.mlw" for credit.** There should be only one configuration file for both objectives. Please submit the malheur configuration file in the Gradescope assignment **Project Malware Analysis - Phase II**.

Reflection:

Now that you have some experience analyzing malware, take a moment to read this brief reflection. For this project you used analysis tools that do the analysis for you. In practice, entire teams of people are devoted to work on a single malware executable at a time to debug it, disassemble it and study its binary, perform static analysis techniques, dynamic analysis techniques, and other techniques to thoroughly understand what the malware is doing. Luckily for you, it takes an enormous amount of time to perfect/improve the skills of malware analysis, so we don't require it for this project. However, to give you a scale of how much work this all takes, consider that antivirus companies receive somewhere on the order of 250,000 samples of (possible) malware every day. We had you analyze 5 binaries. Imagine the types of systems needed to handle this amount of malware and study them thoroughly enough for that day, because the next day they're going to receive 250,000 new samples. If a malware analysis engine is unable to analyze a piece of malware within a day, they've already lost to malware authors. Also consider that not all of the 250,000 samples will be malicious. According to "Prudent Practices for Designing Malware Experiments: Status Quo and Outlook"(Rossow et al., 2012), as many as 3-30% may be benign!

Another way to look at the size issue of malware analysis, consider this paper "Needles in a Haystack: Mining Information from Public Dynamic Analysis Sandboxes for Malware Intelligence."(Graziano et al., 2015) where the authors discovered that notorious malware samples had actually been submitted months, even years before the malware was detected and classified as malicious in the wild.

Remember, analyzing malware is a delicate and potentially dangerous act. Please be cautious and use good practices when analyzing malware in the future. If you let malware run for too long, you may be contributing to the problem and may be contacted by the FBI (and/or other authorities) as a result of this unintentional malicious contribution. At Georgia Tech, researchers, professors, and graduate students are able to analyze malware in controlled environments and have been given permission by the research community to perform these analyses long-term. We make efforts to contact the general research community and Georgia Tech's OIT Department to inform them that we are running malware,

so they won't raise red flags if they detect malicious activity coming out of our analysis servers.

For your curious mind:

There is disagreement in the malware research community as to what exactly classifies malicious activity. For example, some say that adware is a form of malware, while others do not. Can you think of arguments for either side? Let's take this kind of thinking one step further. As a thought experiment, ask yourself this: If a piece of software has malicious code contained within it, but the malicious code is never executed when it is run, is/should that software be considered malicious? What if the malware author intentionally put in a buffer overflow vulnerability that allows someone to execute that malicious code? So, the only way of knowing the code can be executed is to exploit the malware. This seems like it would be a much more advanced form of trigger malware doesn't it? Think of other tricks malware authors may employ to prevent researchers from discovering a malware's true intentions.

Be careful if you ever get your hands-on malware source code. We always make sure we read and fully understand malware source code before we compile and run it. Remember, safety is the number one priority in malware analysis.

If you're interested in reading more information about researching malware, we recommend you read "The Art of Computer Virus Research and Defense" by Peter Szor. It's known in the research community as a must-read for those interested in studying malware.

References

Rossow, C., Dietrich, C. J., Grier, C., Kreibich, C., Paxson, V., Pohlmann, N., . . . Steen, M. V. (2012). Prudent practices for designing malware experiments: Status quo and outlook. 2012 IEEE Symposium on Security and Privacy. doi:10.1109/sp.2012.14 http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=6234405

Graziano, M., Canali, D., Bilge, L., Lanzi, A. & Balzarotti, D. (2015). Needles in a Haystack: Mining Information from Public Dynamic Analysis Sandboxes for Malware Intelligence.. In J. Jung & T. Holz (eds.), USENIX Security Symposium (p./pp. 1057-1072), : USENIX Association. https://www.usenix.org/system/files/conference/usenixsecurity15/sec15-paper-graziano.pdf