- Introduction:

Cybersecurity incidents are at an all time high[1] and the complexity of modern software mixed with the inherent difficulty of the problem of security[2] creates a landscape where it is almost guaranteed that you will have security vulnerabilities in software. While this may seem a bit discouraging there are some remedies to the problem; The most important remediations are isolation[3] and detection however in this project we will focus on just detection.

Having a useful and working detection system for malicious behavior is incredibly useful as it allows cybersecurity professionals to take a closer look at programs or remote computers that may be acting up and respond to them in real time before the attack has been perpetrated.

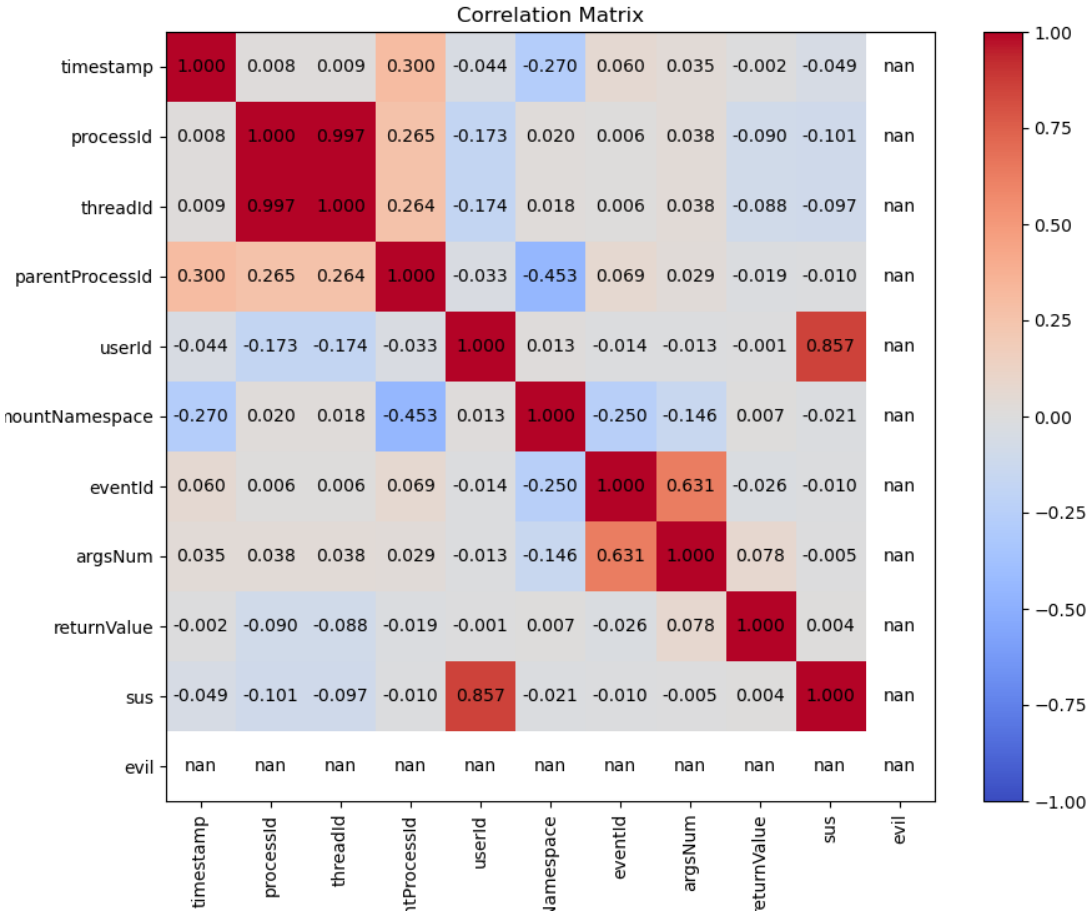
In order to detect malicious behavior we must first define it. This is incredibly challenging and will vary system by system so for our purposes we will use the definitions of malicious and suspicious behavior outlined in the BETH cybersecurity dataset paper. This dataset, hosted on kaggle has over 8 million data points collected on various honeypots through two sensors; sensor one is a network logger and sensor two logs the calls made to the kernel. Due to the combination of more data for the kernel calls and easier data to sort through for this project i opted to use the kernel call sensor

The data is described in the paper as the following

Feature	Type	Description
timestamp	float	Seconds since system boot
processId*	int	Integer label for the process spawning this log
threadId	int	Integer label for the thread spawning this log
parentProcessId*	int	Parent's integer label for the process spawning this log
userId*	int	Login integer ID of user spawning this log
mountNamespace*	int (long)	Set mounting restrictions this process log works within
processName	String	String command executed
hostName	String	Name of host server
eventId*	int	ID for the event generating this log
eventName	String	Name of the event generating this log
argsNum*	int	Length of args
returnValue*	int	Value returned from this event log (usually 0)
stackAddresses	list of int	memory values relevant to the process
args	list of dictionaries	List of arguments passed to this process
sus	int (0 or 1)	Binary label as a suspicious event (1 is suspicious, 0 is not)
evil	int (0 or 1)	Binary as a known malicious event (0 is benign, 1 is not)

1. See heading two https://blog.cloudflare.com/an-august-reading-list-about-online-security-and-2023-attacks-landscape/
2. Testing to see if a user can access some information is easy but testing to make sure no other user can access that information is incredibly difficult
3. https://cyberinsight.co/what-is-the-importance-of-isolation-in-cybersecurity/

While the dataset provides a lot of data a lot of it can be dropped or needs to be simplified. Hostname simply refers to the computer itself and wouldn't change given a malicious kernel call so that can be dropped. The stack address, while it may be useful, is incredibly hard to generalize into a simple model so for our purposes we will be dropping it. The args, while seemingly incredibly useful are very hard to work with without a very complex model which we will not be using so sadly we will have to drop it although we can keep the args num. We can see some other things can be dropped when we do a correlation matrix



Note: there are no evil markings in the training set which is why we get a nan when running the data visualization

Thread ID and process id are incredibly correlated so we can drop thread id.

From here we want to map all remaining values to some binary or lower value number. For the timestamp I ran a standard scaler. The other values are a little

more tricky but we can map them to a binary value such as OS and not OS and /mnt and not slash mount and success/failure (more info is given in the paper on page 6)

Now all that's left to do is code this up using pyspark dataframes and get cloud storage working. Csv files are semi structured data so we will use a datalake on the cloud to store it (note in the code it pull it from github in order to save money on running the datalake 24/7 but i would recommend that you upload it to a datalake and change the links in the first few lines)

Once the data is preprocessed we can run an isolation forest on it using either sus or evil as the label. Sadly mllib does not yet support this so I opted to use the sklearn implementation that is not parallelized. Microsoft does have a package that can do this however it is not open source.

The choice to use isolation forest is because it is very good at predicting anomalous data (such as cybersecurity incidents)

Overall the code was more accurate on the validation then the testing by a significant margin indicating the datasets are too different to be useful

Regardless of the results while the model may have been relatively accurate and useful in detecting suspicious kernel calls on the validation set. what is an isn't suspicious will vary use case by use case so this model will never be effective in a general security situation