Final Report

In today's advancement of using cybersecurity, organizations are facing a constant amount of threats that compromise sensitive information, disrupt operation, and even damage their reputations. With cyber-attacks becoming more frequent and widespread, it is important for organizations to implement comprehensive security systems to protect their digital assets. The objective of this project was to design and develop a comprehensive security system for a hypothetical organization applying the concepts and techniques learned throughout the course, including systems engineering principles, UML modeling, python automation, data analytics, and risk management. We were to utilize UML to model the system architecture as well as processes. Python automation scripts for various cybersecurity tasks and data analytics and machine learning for threat detection and response were to be integrated into the system too. As the final part of the project, risk management and disaster recovery plans were to be developed based off of how the system functions through the diagrams and scripts.

This project mostly aimed to integrate an automated system for log analysis and vulnerability scanning because by automating the identification of malicious activity, security team personnel can focus their efforts on addressing critical issues before they evolve into full-scale attacks. Our assigned organization for this project was a financial organization. Despite having a basic security system in place with a windows defender, our organization ultimately lacked a systematic automated approach to identifying malicious activity and vulnerabilities. As a result, there were many security incidents that needed to be addressed such as access control, incident detection and response, secure DevOps, and a secure encrypted database.

These security incidents were addressed in our system through a variety of tools and techniques. UML or unified modeling language diagrams were used to design and show the system workflow. They helped plan the project, ensuring all security requirements/incidents for the system were taken care of. Python was the programming language used for developing the automation scripts. Its libraries and frameworks, os, subprocess, psutil, smtplib, nmap, and scapy were all used for log analysis, vulnerability scanning, and generating email alerts. GitHub was used for version control and as an easily accessible platform to store

all the documentation and scripts for the entire project. By using GitHub, changes to the project could be easily tracked allowing for excellent collaboration.

The UML diagrams were the first part of the project tackled, being useful for the planning and design of our system. Our system requirements of addressing the security incidents of access control, secure DevOps, a secure encrypted database, incident detection and response, and log analysis were demonstrated in UML use case and activity diagrams. Access control, secure DevOps, and a secure encrypted database are most accurately seen/addressed in the UML diagrams. We solved access control by implementing multifactor authentication (MFA) and a firewall to pass through. We implemented a secure encrypted database to store our data instead of a windows defender and google drive. DevOps are also integrated into the system through the incident response security team, which emphasizes collaboration with the rest of the organization employees and the other external security, internal security, and systems administrator teams.

Although incident detection and response, log analysis, and vulnerability scanning are seen in the UML diagrams, we dive into them further with the implementation of our python scripts. Log analysis became automated in the system through python scripts that filter logs looking for patterns, like unauthorized access and failed login attempts. This automation is key to the system's efficiency in processing datasets. The scripts send alerts to employee's emails based on the predefined thresholds of network traffic and CPU usage through the smtplib library. Vulnerability scanning became automated in the system through the python scripts using nmap, the subprocess library, and scapy. The scripts conduct regular scans of network traffic and vulnerabilities, as well as generate a report that highlights critical vulnerabilities if the network is up or down. This report is reviewed by the security teams to be taken action upon.

Ultimately, this project resulted in the creation of diagrams and scripts that include tools to make the system more secure. The diagrams illustrate the interaction between users and the system by showing that once the employee signs in, their information is stored in the active directory. Once the employee signs in, they are then prompted to set up MFA so that each time they log into the system they have to go through it. After MFA, they will be authenticated to pass through the firewall. Now, the user can make

appointments and do their tasks. When they want to complete their tasks or schedule an appointment the active directory checks if they can. If no the user will be directed back to their tasks page or appointment schedule, if yes then they can input their information into the system. A logging request will be sent to validate the input. If the input is validated, the employee is done with their tasks or scheduled successfully an appointment. If the input is not validated, the incident response process will be initiated.

The scripts of this project successfully parsed logs to identify unusual network activity, thus improving the accuracy of threat detection. They identified suspicious patterns within the logs that were then generated in a summary report text document named summary_report.txt. Nine suspicious logs were found in total. Our python scripts also monitored the aspects of system performance by measuring CPU usage. They collected performance metrics from the system at regular intervals using the psutil library, logged the performance data, and generated an alert via email when the performance threshold of CPU usage of 0.1% was met or exceeded. The percentage usage metrics were documented into a file named performance_log.txt. The scripts implemented an effective automated alert generation system when the CPU usage was high by using the email API sendgrid and the smtplib library. The code sends an email to the specified email address of an employee to alert them that their CPU usage is high. Finally, our python scripts implemented an effective routine to scan for vulnerabilities and monitor network traffic. Nmap, the subprocess library, and scapy are in the automated code to run security checks. By using scapy in our code the packet counts of our code can be seen, 10 packets. By using nmap to scan for vulnerabilities, we can see the network standing.

Through the process of data analysis, zero-day exploits will be common, as there will be attacks on the system that will be unknown to fix at first. Alluding from zero-day exploits, insider threats from our employees could also occur unintentionally. DDoS attacks, overwhelming the system to disrupt services, are another thing our system is susceptible to since the system could have a lot of users trying to make the same appointment or do their tasks together. Data breaches and phishing are risks that are present too. If the active directory is down, there are still measures in place like MFA and the firewall that must be passed through for someone to login. Since there are three measures in place to get into the

system, there is not a critical level for a data breach. There is still a high level for it though if MFA is bypassed. Emails can also be sent from an outside source, as with any company system. Systems admin, the internal security team, and the incident response team all need to be aware of the practice of phishing. Lastly, there is cloud vulnerability in our system. Transferring old data to the new database will make the data more secure for employees to use, but there could be misconfiguration of the data while this transfer is occurring.

In order to best manage the risks and vulnerabilities within the system, a risk management and disaster recovery plan were developed. Preparation is step one; ensuring that employees in the company have proper training to recognize and report security incidents is important with this new system. The internal security team will implement a training program for employees to recognize an incident and show how to not do some of it. Detection is step two; the incident response team will assess the impact of incidents by analyzing and documenting with logging request the employee put through. Response is step three; the incident response team will isolate parts to fix and remove the cause of the incident. Recovery is the last step; the systems admin will restore the affected part of the system. Post incident review will then happen after the system update is validated. All employees and security team members will be advised of the incident that happened and why. The security teams will report on the lessons they learned and their response actions through documentation. Employees will apply what they have read through the documentation to avoid repeatable incidents if applicable.

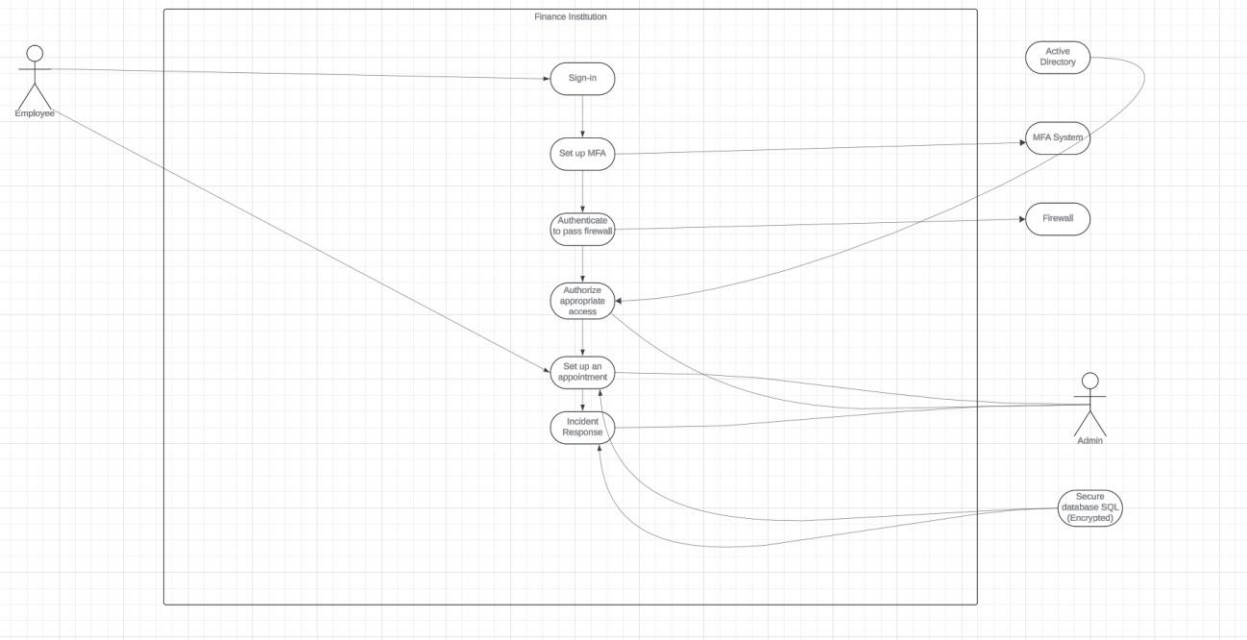
While we had functional scripts and effective automation within our financial cybersecurity system, we had a challenge finding a log to test and put into our code. We found a small dataset to process for the basic data analysis for this project. A bigger dataset would have also been a challenge to integrate because a high volume of financial data can be overwhelming to a system. Since our system is of the financial sector, the data analysis process is very useful since it provides alerts to our employees and security teams when there are potential threats. A trend in the financial sector of cybersecurity is that risks need to be identified and put out, if possible, immediately since money is involved. We also had a difficult time with the python coding at first because we all have minimal experience with it. We are first-

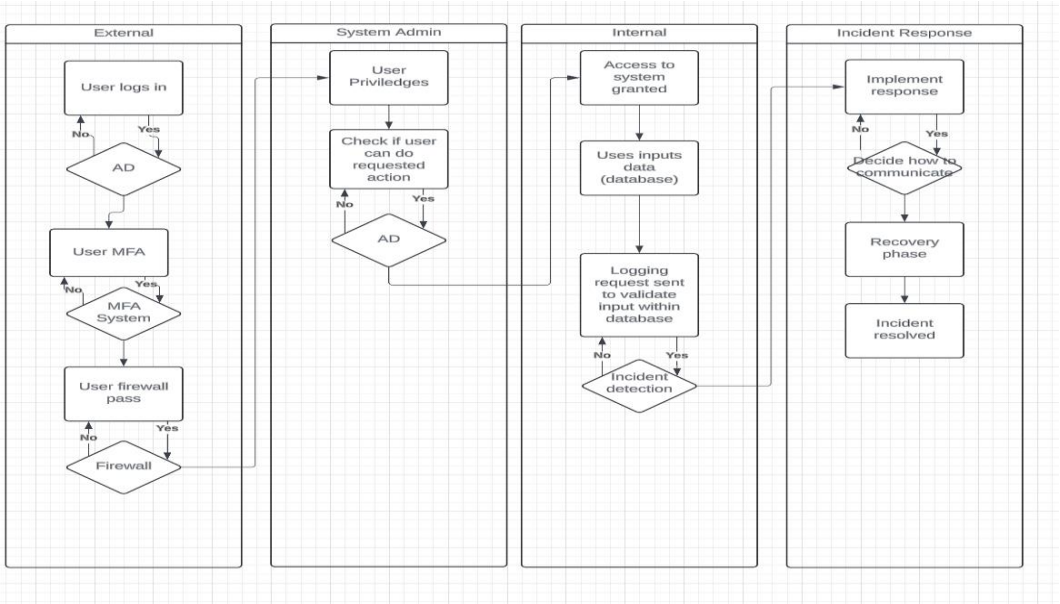
year students at George Mason University who are in their first classes as CYSE majors. A lot of reading, research, and collaboration was needed to get functional diagrams and code.

Even though the system performs well, there could be additional improvements. Integrating machine learning algorithms could help the system learn from its data to be able to raise its ability to distinguish between real user activity. This would reduce the number of potential false incident alerts to the security teams. Since there is full automation when it comes to scanning for vulnerabilities, it would be reasonable to also have the internal security team conduct regular manual audits to ensure the system does not miss any new vulnerabilities. A third recommendation for the system would be to update employee and security training regularly based on the system incidents and vulnerabilities. For instance, recognizing DDoS attacks and phishing are important to know about.

Overall, this project resolved several security challenges within our financial organization. To summarize, these challenges included ineffective log analysis and vulnerability scanning, slow incident response, access control, and ineffective database storage. By automating log analysis and vulnerability scanning especially, the organization is now better able to detect and respond to incidents in a timely manner. Several lessons were learned from group members during this project experience regarding system design and python scripting. The main takeaway, however, is the power of automation over a security system. Automation improves security operations significantly and will continue to be needed in future systems as industries continue to advance.

UML diagrams: Use Case and Activity





Python Script:

```python
import smtplib
from email.message import EmailMessage
import psutil
import subprocess
from scapy.all import *
from email.mime.text import MIMEText
import os
from sendgrid import SendGridAPIClient
from sendgrid.helpers.mail import Mail

with open('C:\\Users\\srius\\Downloads\\HDFS_v1\\HDFS.log', 'r') as file:
  logs = file.readlines() #The log file we found online that we are importing

suspicious_logs = [log for log in logs if 'failed' in log.lower() or 'unauthorized' in log.lower()] #Scans for suspicious logs in our file

# with open('summary_report.txt', 'w') as f:
f = open('summary_report.txt', 'w') #opens summary_report.txt
f.write(f"Total suspicious logs found: {len(suspicious_logs)}\n") #Lists the number of suspicious logs found
for log in suspicious_logs: #Lists the suspicious logs
  f.write(log + '\n')
f.close #Closes summary_report.txt


#Get CPU Usage
cpu_usage = psutil.cpu_percent(interval=1) #Uses psutil to find the CPU usage as a percent
print(f"CPU Usage: {cpu_usage}%")

#Get Memory Usage
memory_info = psutil.virtual_memory() #Uses psutil to find Memory usage as a percent
print(f"Memory Usage: {memory_info.percent}%")

with open('performance_log.txt', 'a') as f: #Opens performance_log.txt
  f.write(f"CPU: {cpu_usage}%, Memory: {memory_info.percent}%\n") #Writes CPU and memory usage in performance_log.txt

if cpu_usage > 0.1:
  print("ALERT: High CPU usage detected!") #Outputs alert if CPU usage is > 0.1


def send_alert(subject, body): #defining function to send email
  msg = EmailMessage()
  msg.set_content(body)
  msg['Subject'] = subject
  msg['From'] = 'sriusa.annamalai@gmail.com' #The email the alert is being sent from
  msg['To'] = 'srideviannamalai02@gmail.com' #The email the alert is being sent to

  with smtplib.SMTP_SSL('smtp.sendgrid.net', 465) as smtp:
    smtp.login('apikey', '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') #Using Email API to send alert

    try:
      smtp.send_message(msg)
    except Exception as e:
      print(e)

#Example: Send an alert when CPU usage is > 0.1
if cpu_usage > 0.1:
  send_alert('High CPU Usage Alert', f'CPU usage is {cpu_usage}%')


def run_nmap(target): #Defining function: run_nmap
  result = subprocess.run(['nmap', '-sV', target], capture_output=True, text=True) #Uses subprocess to scan for vulnerabilities
  print(result.stdout)

run_nmap('127.0.0.1') #scan local host

def monitor_packets(pkt): #Defining function: monitor_packets
  if pkt.haslayer(TCP) and pkt.haslayer(IP):
    print(f"Source IP: {pkt[IP].src}, Destination IP: {pkt[IP].dst}")

sniff(prn=monitor_packets, count=10) #Capture 10 packets
```

Folder Links for Easier Viewing:

https://gmuedu-my.sharepoint.com/:t:/g/personal/gashton2_gmu_edu/Ea-TXwyunaUg0BCb5fC2scB5Y7Xu_5ElPlxnwqbCv6YdQ?e=Igapjl

https://gmuedu-my.sharepoint.com/:t:/g/personal/gashton2_gmu_edu/EfaHval2-ZxMhQPnMOdXtcIBKrPsa4ZQgHGmDV5zmawCVQ?e=PwnAkX