AwesomeDEP

Submitted by

zID: z5492132

Name: Aman Sharma

1. Overview and Research

Data exfiltration prevention is critical in protecting sensitive information from unauthorised access and theft. The goal of this project is to develop and implement methods to detect, prevent, and mitigate data exfiltration, ensuring that sensitive data stays within the organisation's control.

1.1 Why did we choose to do this?

Implementing a data exfiltration prevention (DEP) solution is a vital step toward data security. A DEP solution is meant to monitor, detect, and block unwanted attempts to transmit sensitive data off of a network. It has various benefits:

- DEP solutions prevent sensitive information, such as intellectual property, personal data, and financial records, from leaking. This is necessary to comply with data protection requirements such as GDPR, CCPA, and HIPAA, which require enterprises to protect private data.
- DEP solutions can detect and prohibit unwanted data transfers, lowering the risk of insider threats such as workers or contractors attempting to move sensitive data outside the firm, whether on purpose or by accident.
- DEP solution systems use real-time monitoring to detect and prevent data exfiltration attempts as they occur, allowing enterprises to respond quickly. This mitigates the possible effect of breaches and aids in data protection.

Furthermore, we made a simplified DEP solution as there is a gap in the market for DEP solutions for small businesses and individual users. This was made very apparent upon further research.

DEP solutions are widely available in the market for larger companies, however for individual users there are very limited solutions when in reality they also share similar risk to the large companies when it comes to losing sensitive personal data.

1.2 Problem Statement

Design a Data Exfiltration Prevention (DEP) solution that is affordable and easy to use for both individual users and small to medium-sized businesses. This will help close the security gap between big businesses and smaller organisations by giving them a strong defence against the loss of sensitive personal data and cyber threats. Overall, the goal is to develop a product that is user friendly, open-source, and scalable with current and upcoming industry standards.

1.3 Research On Existing Products

After extensive investigation, OpenDLP and Snort were found to be the most recommended services. These services were primary competitors to our product. However, these services came with a range of differing benefits and critical flaws.

1.3.1 Research on OpenDLP

OpenDLP is a great product that has now just failed to hold the test of time and newer security threats. It seemed very promising in the preliminary stages as it was completely open source with the entire code base being on GitHub. This seemed to solve one of the set goals, having a cheaper alternative to data security for small businesses and individual users.
OpenDLP is very customizable, users can modify the tool to make it fit for their own security needs. This may mean modifying the program to fit certain network environments, adding new rules, or changing the scanning parameters. This will be useful for the very informed user, however this degree of freedom for the uninformed user is detrimental. The uninformed user being the majority of the market this would be a major issue to use OpenDLP.
Then there is the fact that OpenDLP is widely considered outdated now. Given it was a popular open-source option for data discovery and basic data loss prevention, it has not seen consistent or any major recent updates, and several factors contribute to its being largely obsolete. These are major security concerns leading to OpenDLP not being a very useful tool in today’s market.
However, based on its relative functionality for the individual user it can be used for comparative analysis to the newly implemented solution.

1.3.2 Research on SNORT

For real-time traffic analysis and packet logging, Snort is a well-known intrusion detection and prevention system (IDS/IPS). Its strong feature set, scalability, and adaptability make it highly respected in network security. It is well regarded in the marketplace as being a product that is effective, meets industry standards and is receiving relatively consistent updates.
Despite having very powerful features, Snort is a small (light weight) product and can be set up to function efficiently on low-resource computers. This degree of user consideration is great especially as it will eliminate the need for large hardware investments and enables simple deployment in a variety of network situations, and

lower the total electronic waste a company has when changing to implement SNORT for their business.
However, it is not free for small to medium size companies. In the scope of the problem statement it is established that the ideal DEP solution should fit into three simple categories:

- Be user friendly,
- Ideally open-source,
- And continually improved, such that it will be able to face new security threats as they come.

Snort, though a very effective and useful program, is licensed and in many cases can be out of reach for many small to medium sized businesses.
That being said, the goal of this project is to try to implement a very similar functioning open-source program that is user friendly and scalable for any DEP security needs. Thus, we will use SNORT, similarly to OpenDLP, for a comparative analysis, essentially to see where this project stands compared to these larger industry ready tools.

2. Learnings and Key Findings

There was a serious gap in our knowledge when it came to networks and network analysis. This was a slow and lengthy process to address the gaps in our knowledge that needed to be corrected by the time we needed to start implementing the project in code. The links to the resources we used to learn are listed as footnotes.

2.1.1 Learning Wireshark

To mitigate this we took a large amount of time to learn the operations of WireShark, a very useful tool that can track network packets. The use of this would be key for the project, we mainly used Pyshark for the preliminary implementation. Pyshark is a powerful Python wrapper for Tshark, the command-line utility of Wireshark. With the use of Wireshark's official documentation (mainly their learning page[1] and documentation[2]) and other resources (mainly youtube videos[3]). The youtube videos were a key resource that gave a lot of information about Wireshark's usage from basic functionality to extracting certain packets based on given conditions. While this gave us a great deal of information we had to test it thoroughly and the tool that was a game changer here was TryHackMe. There were comprehensive guides and challenges that we could follow to much better understand how to use Tshark[4]. Once we completed the challenges and learned more about the Tshark wrapper for python Pyshark we could begin making attempts at deep network traffic analysis.

2.1.2 Benefits and Limitations of Wireshark

Wireshark is great at in-depth packet analysis and packet visualisation. This would work well for many smaller implementations of simple DEP's. However, it comes with some significant drawbacks. It does not allow the creation of packets. Wireshark has no ability to manipulate packets; it is only for analysis. Although its command-line equivalent, Tshark, has some scripting support, Wireshark's automation capabilities are more constrained. The GUI is used for the majority of operations, which might be laborious for repetitive tests.

[1] Wireshark Learn - https://www.wireshark.org/learn
[2] Wireshark Documentation - https://www.wireshark.org/docs/
[3] Wireshark Masterclass with Chris Greer - https://www.youtube.com/playlist?list=PLW8bTPfXNGdC5Co0VnBK1yVzAwSSphzpJ
[4] TryHackMe - https://tryhackme.com/module/network-security-and-traffic-analysis

2.2.1 Learning Scapy

Scapy lets users to programmatically create, send, and manipulate packets, which is useful for activities like penetration testing, custom packet production, and traffic simulation. Designed to be Python scriptable, allowing for fully automated network operations. This makes it ideal for complicated testing scenarios, automated scans, and assault simulations. You can use Scapy in scripts to assess network resilience, generate packet storms, and stress-test applications. We mainly used the official Scapy documentation[5] for learning and the main functionalities. A thorough understanding of Wireshark and Tshark helped us learn Scapy more easily. Wireshark (Tshark) provides you with a solid foundation in packet structure, network traffic patterns, and protocol intricacies that are directly applicable to Scapy jobs. This took a lot of testing with various attempts with different functions like "sniff" and other very useful functions like but not limited to "get_if_list", "conf". There was a definite increase in difficulty in using Scapy as its versatility was also a relative drawback with too many possible features for similar functions. Furthermore, Scapy being a purely script-driven, which may be a disadvantage for individuals who prefer visual analysis. It requires a more complex understanding of Python and command-line interfaces.

2.2.2 Scapy Vs Wireshark

Given that Scapy is great for packet generation, it's trickier to use for capturing and analysing live traffic without the assistance of other tools. Wireshark has some automation capabilities, it is not nearly as flexible as Scapy for creating complex scripting or simulations. Scapy is known for its versatility and automation capabilities. Thus, Scapy is a better choice if you need to produce, alter, or simulate packets in addition to analysing them. Its interface with Python allows for extensive scripting and automation, which can significantly expedite repetitive or complex procedures. While Wireshark excels at packet analysis, Scapy's active capabilities make it a more adaptable and successful tool in dynamic networking circumstances or security jobs that need packet production and engagement.

[5] Scapy Documentation - https://scapy.readthedocs.io/en/latest/

2.3 Front-End Design, DataBases, and Flask in Python

Before undertaking this project I had little to no experience with front-end web development. This learning was mainly done by watching a multitude of different tutorials on youtube. The freeCodeCamp's full tutorial food beginners[6] was by far the most helpful. Using this I created a rather basic website, that is well coloured and shows the desired output we wanted in our product. This was an interesting challenge. Currently taking a back-end course and having done a course and a lot of work on databases the connection part was rather simple to understand and implement in python. The main resource I used for the database component was freeCodeCamp's SQLite3 with python video[7]. That was a very helpful resource to refresh my understanding of databases. For flask I, again, used freeCodeCamp's Flask for python youtube video[8].

2.4 PF Rules

These are a set of tools that work only on macOS. PF rules allow us to control how data packets are handled. My main use case for them was blocking out all leaving network traffic. This took quite a bit of testing and was dangerous at the start with having some rules that set off my wifi until a reboot was done. There are multiple downsides to using PF rules, but for macOS it was the most efficient way of implementing the project. The drawbacks are mainly with OS configurations as PF does not work on Windows and with the consideration that we did not have a Windows device to test the product on we chose to continue with PF rules. The main learning resource for this was the manual pages, using 'man pfctl', and testing it out till there was a working product.

[6] Learn HTML - Full Tutorial ~ freeCodeCamp https://www.youtube.com/watch?v=kUMe1FH4CHE&list=PLWKjhJtqVAbnSe1qUNMG7AbPmjIG54u88
[7] SQLite3 Databases with Python ~ freeCodeCamp https://www.youtube.com/watch?v=byHcYRpMgI4&t=712s
[8]Flask for Python ~ freeCodeCamp https://www.youtube.com/watch?v=Z1RJmh_OqeA

2.5 General Python Libraries - Relearning

We quickly came to the realisation that we had to upskill in many other minor aspects of system understanding in python to better be able to implement the DEP solution. One of the key libraries was os, this was key for us as we needed it in many different ways from getting an environment to many others. We also used the time library for getting timestamps. Lastly, to make sure the password was securely inputted we used getpass. There were also a few other libraries, these were just the ones I most frequently used. Re-familiarising with these libraries or learning them proved relatively little challenge.

3. Implementation

The goal being an implementation that will achieve complete DEP functionality whilst still being user friendly. This was a slow and lengthy process to get to the end product we have now. There were many changes made from a very simple implementation using Pyshark and a bare bone limiter that would log into the terminal. That all slowly changed to the end product we have now that displays a web dashboard that shows the suspicious packets found more accurately with geolocation and timestamp features. The main drawback to this implementation is that it would only work to its fullest capabilities on macOS. This is mainly due to the fact that we do not have a windows machine to test it on, hence we have made a working macOs based product.

3.1 Main.py

The main purpose of this file is to capture packets, then process them to see if they are suspicious or not. If they are deemed suspicious then we log them into the network monitor database. In this file there are a couple of different files that run for easier usage for the end user. Essentially, the main file takes all of the files used in the implementation and runs them all in one place. The process packet function works in main to flag an alert system that will send an os level alert and block all outgoing network traffic. My main involvement with main.py was with debugging and fixing some of the requirements files such that main worked more effectively.

3.2.1 Limiter.py

Here I have created a class called limiter that can be easily called in main.py. The main functionality of this file is to block the outgoing network traffic. The implementation has two special passwords, one that will prompt the user to enter every time it is called, as per main's functionality it was every time that a suspicious packet is found network traffic is blocked and the user is asked for the override password that will allow them to regain the internet password. Usually there are a multitude of packets so there is an admin level password, if that is entered then it will ignore all the packers, do note if the admin password is used then most of the packets are not logged as it will exit the program itself and only log the first packet found. This is an ease of use feature we added during tests.

3.2.2 Challenges with Limiter.py

There were a multitude of different challenges with the implementation of the limiter. One of the main issues with using PF rules to block out all outgoing network traffic was very risky at times in the preliminary stages. This was mainly due to it blocking our wifi setting, though this was the desired feature without the admin or override passwords in place we had to reboot our systems a couple of times. Given a user can be in front of other people when the passwords are entered, we changed the implementation to use getpass which is far more secure for password handling than simple user input.

3.3.1 Database and Web Dashboard

The database.py file has two main functions, init_db and log_packet. These work very very intuitively. The database is initialised with init_db and a table is created that takes into account the source ip and the destination ip of the packets captured along with a timestamp. Then there is the log_packet function that logs all the packets that are deemed suspicious. This was all very easy to setup and use as we did it with SQLite3 in python and used some database visualisation tools when needed for debugging. The web_dashboard.py file looks into the database with the get_packets function and renders the website in index mainly using the flask function render template. The web dashboard works on local host and is made to run indefinitely updating when new packets are found. The reasoning for working on a local host is again for security reasons as setting a permanent website that generated tokens for users would have more possible web vulnerabilities for us to deal with.

3.3.2 Challenges with Web Dashboard and Database

The main challenges here was getting the look of the website correct, this took a lot of debugging and at one stage we had some database issues that rendered the website rather useless. Fixing that was a major debugging challenge, hence the reason for inspect_db.py to be created. That with the help of some database visualisation tools helped fix the errors in the database. Working with HTML was also a challenge as I have not done any prior work with it, hence the website being rather basic, it does however show the user the necessary information in an efficient manner.

4. Conclusion and Review

Overall, this project gave me a lot to learn and think about. In times it was frustrating debugging for hours, but it gave us a collective appreciation of how complex versions of DEP solutions are in general.
For the comparative analysis of our Awesome DEP vs OpenDLP. OpenDLP is expected to provide more complete, cross-platform DLP capability, including network monitoring and centralised management. However, AwesomeDEP is more agile and adjustable for macOS-specific purposes, particularly for simple uninformed users it will be easier to use. Given the fact that our AwesomeDEP will be continually improved it is fair to say given more time to implement more features that will help battle the new vulnerabilities that come with time. This is something that OpenDLP is seriously lacking in.
For the comparative analysis of SNORT vs the AwesomeDEP. SNORT out performs our product in many ways. It is a more full and comprehensive product. Again, for the uniformed user that is wanting a simple product that manages their DEP needs from single users to small companies, our DEP solution would be a thoughtful consideration.