École Polytechnique Fédérale de Lausanne

Research and development of a DLP rules analyzer based on a Cybersecurity framework

by Pierre Colson

Master Thesis

Approved by the Examining Committee:

Prof. Dr. sc. ETH Mathias Payer
Thesis Advisor

Stéphanie Fatelo
External Expert

EPFL IC IINFCOM HEXHIVE
BC 160 (Bâtiment BC)
Station 14
CH-1015 Lausanne

February 28, 2024

Acknowledgments

I extend my sincere thanks to my colleagues Clément and Julien for their assistance and camaraderie throughout this project. Their collaboration has been a cornerstone of my daily research experience.

I am indebted to Stéphanie, my company supervisor, whose insight and guidance have been critical to the progression and success of my work. Her expertise has been a great asset.

Moreover, I would like to express my gratitude to Mathias Payer, my academic supervisor from EPFL, for his constructive criticism and invaluable feedback on this undertaking.

I am also thankful to EPFL for providing an enriching academic environment, and to the company for the practical support and the opportunity to conduct this meaningful research.

I must acknowledge HexHive for their master thesis template, which was instrumental in the structuring and formatting of this report.

Lastly, I wish to thank my family and friends for their unwavering support, which has been a continual source of motivation.

Lausanne, February 28, 2024 Pierre Colson

Abstract

This report focuses on managing complex Data Loss Prevention (DLP) rules to prevent sensitive data leaks in a corporate environment. Existing DLP systems lack sufficient tools for rule optimization, leading to potential inefficiencies and security gaps. As security requirements increase, DLP rules become more and more complex, increasing the risk of rule misconfiguration and thus the risk of data leakage. The project aims to introduce a tool designed to analyse and optimise DLP rule sets. Such tools are not yet available for DLPs, although they are common for firewall policies. The project establishes a parallel between DLP and firewall, and adapts and completes some firewall designs for DLPs, such as anomaly detection or rule reordering. Additionally, the analyzer assists administrators when inserting new rules into DLP policies, and establishes a cybersecurity coverage score for a given configuration. By generalising DLP rules, formalasing DLP conditions, this tool detects redundant and conflicting rules, suggests optimisation strategies to improve computer performance, and aids in the management of new rule additions. The analyzer generates visualizations and rule summaries, making policies more readable. It is already used and deployed in the company's environment, and provides meaningful feedback. The evaluation indicates that the new rule configuration leads to a stronger security posture and better coverage against data leak. Experiments show that performance improvement is not heavily dependent on rule order, but this one remains crucial in enforcing security requirements

Contents

Chapter 1

Introduction

Why is Google's search algorithm so effective? Only a select few people know the answer to that question. In fact, Google keeps this algorithm a closely guarded secret, and its value is likely immeasurable. The same applies to Coca-Cola's recipe or The New York Times's definition of a bestseller. These pieces of information are known as intellectual property. Leaking intellectual property can result in significant costs for an organization. Companies commonly process personal data from both customers and employees, which is sensitive information protected by various regulations. Protecting intellectual property, and sensitive information has become a significant challenge for companies.

To prevent sensitive data leakage, companies must to identify this data. The identification process can be automated using pattern recognition or machine learning, for example. However, company-specific sensitive data may not be easily detectable using these techniques. Intellectual property and sensitive data can also exist in non-textual formats, such as images, videos, audio, which greatly complicates any automatic detection and classification. Defining how sensitive an organization's data is often left to those who work with it. Once defined, the sensitivity level, should be attached to the corresponding file, in order to allow tools or softwares to monitor the use of this data. This classification is necessary to prevent data leakage and is used by DLP software.

Data loss prevention (DLP) software and tools are used to monitor and control endpoint and server activities, filter data streams on corporate networks, and monitor data in the cloud to protect data at rest, in motion, and in use [9]. Most DLP solutions offer tools to discover and define the sensitivity of enterprise data, monitor the use of sensitive data, and prevent sensitive data from leaving the company. This thesis focuses on DLP endpoint. A DLP endpoint is a software agent installed on a user's laptop. This software captures and analyzes system operations, such as file copies, file moves, network operations, file updloads, and copy/paste actions. These operations are always based either a file or data from a file. The sensitiviy level, called classificaiton of the file/data, is attached to the file. The endpoint can monitor or block an operation if it does not follow the security requirements defined by the company.

Not all companies have the same security requirements. For example, a bank or a governement entity is expected to be more stingent and cautious about data leakage than a car garage. Each company defines security requirements based on a defined classification level. Such requirements may enforce encryption for certain files, limit access for others, or have no specific constraints for public files. These requirements serve as guidelines and should be enforced using software. DLP is one of the numerous tools that can be used to enforce security requirements. By default, DLP endpoints simply inspect system operations but administrators need to write DLP rules to block some of them. A DLP rule is an implementation of one or several security requirements. As security requirements increase in size and complexity, so do DLP rules. Managing rules to prevent data leakage can be a challenging task particularly when companies's requirements are complex. This thesis aims to develop solutions to assist administrators in managing DLP rules to reinforce the coverage of their security requirements.

Currently, there are various ways in which data can leak from on computers, including through communication systems like email or direct messaging, as well as through web browsing, cloud solutions, NAS, and external drives. Data Loss Prevention (DLP) solutions were introduced to the market in the 2000s, and have since been continuously improved to extend their coverage. The primary objective of existing solutions is to enhance data leak coverage and stability while minimizing their performance impact. Currently, optimising the rules set up by administrators is not a priority. However, the efficiency of a DLP solution depends on the rules that are created, which are the core of what and how DLP software enforces data loss prevention. Unfortunately, DLP solutions do not have a built-in rule analyzer or optimizer, nor is there an external tool that aimed at improving DLP rule sets, although such tools are common in the firewall world. As DLP solutions become more prevalent, the need for a DLP analyzer will grow.

To configure a DLP, administrators need to add, remove, or modify rules in the DLP rule set. Even experienced administrators find it challenging to compare new rules with all existing rules and identify redundancies and gaps in requirements. The chances of finding and removing redundant rules are further reduced if DLP administrators change. Modifying DLP policies increases the risk of introducing some errors that could make the computer vulnerable to serious data leakage. To mitigate this risk, a method for automatically detecting anomalies in a rule set has been created. This method raises warnings when two rules overlap or when one is shadowed by another, which helps in prevent configuration mistakes.

Computer performance can be heavily impacted by the efficiency of the DLP. This is due to the fact that the entire rule set must be evaluated for each system operation inspected. As policies become larger and more complex, the process of sequentially evaluating conditions of policy rules until the first match is found becomes time-consuming. Therefore, it is important to determine the appropriate action for system operations as quickly as possible. One approach to achieve this, one approach is to reorder rules, and assign lower priority to those that are more likely to be triggered, as they will be evaluated first. As the number and complexity of rules increase, creating new rules and removing existing ones becomes more challenging. The 'Policy Advisor' aims to assist administrators in adding new rules and ensuring that there are no unintended data leak opportunities.

The objective of this project is to develop a tool or software capable of analysing and optimising a DLP rule set. The four main goals of the analyzer are detailed in this paragraph. The first goal is to detect unnecessary rules. Detecting these rules can either simplify the DLP rule set if removed, or draw attention to a misconfiguration. This situation cannot be left unnoticed as it reflects a administrator oversight. The second main goal is to provide a new rule order to theoretically improve endpoint performance. This rule order will attempt to place the more common rules in the first positions of the rule set. It is important to note that rules in a DLP rule set cannot be moved as administrators please, as their order is used to enforce the company's security requirements. The third main goal is to offer suggestions to administrators when inserting a new rule. Once the current rule set is free of anomalies and is performing efficiently, the aim is to maintian it in that state. Adding new rules or modifying existing ones a recurring challenge. The analyzer will provide priority suggestions for new or edited rules to ensure that they do not introduce new anomalies. This will guarantee that the current DLP coverage will not be reduced, due to rule modifications, in the future. The ultimate goal is to establish a high level coverage score of a given DLP rule set. As mentioned earlier, the purpose of the DLP is to enforce security requirements, which are often based on security frameworks, such as MITRE ATT&CK [1]. The analyzer will provide a summary of data types, and exfiltration methods that the DLP currently covers. This summary is crucial in managing how security requirements are met. The DLP does not need to cover every attack or risk vector, but it should have a clear and easily accessible coverage. Any data types not covered by the current DLP configuration should be addressed by other tools or by future DLP rules. The cybersecurity score indicates a coverage rate of the DLP rule set based on predefined security requirements. It is important that the tool is not overly specific to a particular DLP solution, and can be easily adapted to other DLP solutions.

Section 4.1 outlines three different types of anomalies detected by the analyzer. Two of them correspond to rules that cannot be triggered in the given rule set, either because other existing rules already cover the use case for which they were designed, or because some previous rules already catch all the events the anomaly was designed for. The last anomaly reflects a direct DLP misconfiguration since it highlights two rules that have an identical priority number. When the analyzer detects an anomaly, it attempts to re-insert the rule into the rule set, as the anomaly is likely due to an incorrect placement of the rule. Additionally, the analyzer can also suggest priorities and actions that a rule should follow to be inserted into the policy without creating any new anomalies. The analyzer generates a new rule order based on the hit probabilities of each rule, which theoretically improves the performance of DLP. The analyzer has been designed to be easily integrated with other DLP solutions. DLP rules have been standardised, along with conditions they are composed of. One component of the analyzer is reponsible for parsing and converting DLP rules from a proprietary format to a generic representation.

The analyzer has already been deployed in the company's workflow and its features are valuable to administrators. Anomaly detection helped identify unnecessary conditions. The generic presentation of each rule and conditon has also been worthwhile and integrated into the company's rule documentation. This generic representation is now the starting point for each modification in the rule set. It is more concise, readable, and accessible than using the DLP configuration system. A new rule order has also been generated, outlining interesting priority assignments. However, this order is not implemented in production since it reduces the readability of DLP policies and its performance impact is not significant. The order in which rules are organised is largely reponsible for the coverage of security requirements. While the new order maintains the DLP coverage and theoretically improves performance, it reduces the administrator's ability to understand how the order still enforces security requirements. A less practical order can increase the risk of a rule misconfiguration and is therefore not desired. The impact on performance of an anomaly, as well as the improvement offered by the new rule order has been attempted to be measured. However, the test environment remains noisy and imprecise despite the implemented isolation step. No significant performance difference has been measured when the matching rule is in the first position compared to when it is in the last position (35th).

The cybersecurity score computed by the analyzer can determine the loss vectors covered by the DLP and identify which exfiltration means still require addressing by the DLP or another software, such as a firewall or antivirus. For example, the analyzer can reveal that archive data are not monitored or that data exfiltration over Bluetooth is not covered either. This feedback is valuable as it provides a objective evaluation of how well the DLP enforces the defined security requirements.

The analyzer provides a significant workflow improvement in managing DLP solutions. It serves as a complementary tool to the existing DLP configuration console, providing administrators and the company with increased confidence and visibility regarding their security requirements coverage. In addition to these pratical improvements, this work represent a preliminary step towards analysing and optimising DLP rules. The thesis proposes several techniques and methods, inspired by those commonly used in firewall environments, and explains how this knowledge can be applied to DLPs. It highlights challenges that are specific to DLPs and offers a generic representation of DLP rules. The implementation revealed some limitations, and the performance impact of rule placement can have could not be precisely measured. This work reflects what administrators are missing in existing DLP solutions in order to prevent misconfiguration. The analyzer's contribution of offering a coverage score based on a cybersecurity framework is also significant. This tool emphasises the importance of preventing misconfigurations and provides techniques for solving DLP-specific challenges.

Chapter 2

Background

Classification is the process of determining the sensitivity level of data. The sensitivity level represent the potential impact that a data breach could have on the company. For example, the leakage of company's intellectual property would significantly harm the company. It is important to note that not all data poses a commercial risk to company; some of it may contain personal information about employees or customers. Personal data, such as medical information and credit card numbers, must be protected and never leaked outside the controlled environment of the company. File classification can be based on three distinct parts. One way to classify sensitive files is through content-based classification, which identifies the sensitivity level based on its content. This can be done automatically using software that recognizes patterns or fingerprints for example. The second type of classification can be based on context. The context of a file is its location, its file type, it's access right, and so on. For instance, companies may store personal data or any sensitive information in specific predefined folders. This type of classification can also be automated once specific criteria have been defined. The final way to classify a document is user-based. This method cannot be automated, but it allows for more precise classification. Users are responsible for classifying the data they create or use, as they are best positioned to identify the potential risk of a breach. To classify documents, users must be aware of the security requirements of their company. This technique may also lead to over classification. Some users may prefer to over-classify rather than under-classify.

The file's classification should remain attached to that it. The be easily understandable and accessible. This is typically achieved through the use of tags. For instance, one company may have three different classification tags, with the first being "public". This tag would be used for documents that are accessible to the entire company, including those accessible from outside the company. These documents do not pose any risk if case they are leaked outside of the company. The second tag, which represents a file that should be restricted to a specific team within the company, could be labelled as "restricted". Restricted data may include technical documentations and configurations that are not useful outside of the scope of the team. Therefore, it should not leak out of the company, but if it does, it would not substantially harm the company. Personal information and intellectual property should be labelled as “secret“. If secret data were to become available outside of the company, it would pose a significant commercial risk. Encryption may be necessary for this data and access management should be restricted to a select few individuals.

A DLP solution can be utilised to discover and classify an organization’s data. Data classification works by injecting metadata into the data that needs to be classified. This metadata can include the sensitivity level, file format, author, creation dat, and any other relevant information that can be used by the DLP. On Windows systems, the classification information can be stored, for example, in the Alternate Data Stream (ADS). ADS is a method for storing file attributes on an NTFS file system. On UNIX systems, classification information can be stored via extended attributes. The advantage of storing metadata in this way is that it is not interpreted by the file system.

DLP software uses classification tags to enforce security requirements. A DLP configuration is organised thtough to polices, rules, and conditions. Conditions apply specific constraints on certain parts of the system operation. For instance, one condition can enforce that the source file path is from the desktop, while another one could specify that the classification tag must be ’confidential’. Conditions can specify various aspect of a system operation, such as the parent process, operation type, operation destination, and classification tag. A rule consists of a group of conditions that allow it to catch all system operation that meet some specific requirements. For example, a rule can be created to catch all confidential uploads to a cloud service. If all the conditions of a rule are met, it will perform an action. Rule can perform various actins such looging the operation, prompting for confirmation, issuing a warning, and encrypting a file, or simply blocking or continuing the operation. These actions can be grouped together as either ’block’ or ’continue’. Rules are organized into policies, which dministrators can then assign policies specific computer or group of computer. These computer groups can be dynamic and based on operations systems for example. Therefore, policies can be tailored to the needs of certain team within company. Several rules may share a group of conditions. In such cases, these conditions can be placed into a component rule. Component lists are a sets of elements that can be used by any condition. Tor example, a component list can define all file extensions corresponding to an archive document.

This fine-grained control allows the company to have team-specific requirements. Security requirements are defined at a high level, but some team may require stricter guidelines. When applicable, specific requirements are defined for their use cases, which are then implemented and transformed into DLP rules. These rules will then form a new policy that is pushed to computers requiring these specifications.

To make a decision about an operation, rules are evaluated one by one until the first rule whose conditions are met is found. The operation undergoes the corresponding action, and the policy’s remaining rules are not assessed. It is important to evaluate these rules in a specific order, as some may have interactions and require evaluation before others to enforce security requirements. Each rule is assigned a priority mumber that determines the order in which the entire set of rules is evaluated. Rule with lower priority numbers are evaluated first.

DLP solutions are typically divided into three types. The first type, known as networked DLP, is responsible for preventing any network-based data leakage. Network DLPs are situated between employees' computers and the outside network. A DLP system, monitors network events, inspects their content, and based on its classification, either prevents or allows the event. Another type of DLP solution operated at a cloud level. Cloud Access Security Broker (CASB) DLP ensures that data stored online via cloud services will not leak or will be accessed by an unauthorised user. This cloud solution is also based on the classification applied to this data and is typically installed at the cloud level to perform network analysis. The final type of DLP solution is the endpoint solution. Endpoint DLPs are installed on employees' computers, and intercept users' actions and system operations. The analysis is performed locally, and the DLP can monitor apps, email services, print operations, file move to external drives, direct messaging, and more.

This thesis focuses on endpoint DLP, which is a software installed on a computer that can intercept system operations. The software locally matches the operation to a specific rule and with its privileges either blocks or allows the operation. Endpoint DLP is configured through a centralized management console, which DLP administrators use to define rules and policies. Once defined, these rules can then be pushed to company's computers. The communication between the management console and the agent is always initiated by the agent itself. The agent can request for updates on a regular basis, as defined during the installation process, or through a small software. Additionally, the agent communicates with the management console to send log information about which rules were triggered. The management console allows for various action such as restarting an agent, turning it off, disabling specific polices, pushing new rules or requesting for a complete diagnosis for example. Figure 2.1 illustrates the process and the interaction between DLP and system operations.

To better understand the project, it is useful to think of DLP as a firewall for system operations. Both DLP and firewall software are organised around policies, which contain multiple rules. These rules consists of several conditions and a corresponding action. If all conditions of a rule evaluate to true, the corresponding action is taken. In Raleigh, the action can either be 'block' (the operation/the packet is blocked) or 'continue' (the operation/the bracket is allowed). Similar to DLP, in firewalls assign a priority number to each rule to determine the order in which they are evaluated. The software applies the action of the first matching rule to the operation/packet.

Standard firewall rules consist of several fields. Typically these are: Protocol, Source IP, Source Port, Destination IP, and Destination Port. Each firewall rule contains a condition for each one of these fields. This behaviour is different from DLP rules. DLP rules have many more fields available, for example, we can have conditions based on the system operation, the source file path, the destination file path, the process that triggered the operation, the classification tag, the destination website, and so on. The variety of fields means that DLP conditions cannot be standardised, for example, checking a source file path, is completely different from specifying some classification tags. Firewall conditions, on the other hand, are often simply represented by a range of values (e.g. allow packet if destination port is in 80, 8080, 22). In addition, each DLP rule does not need to specify

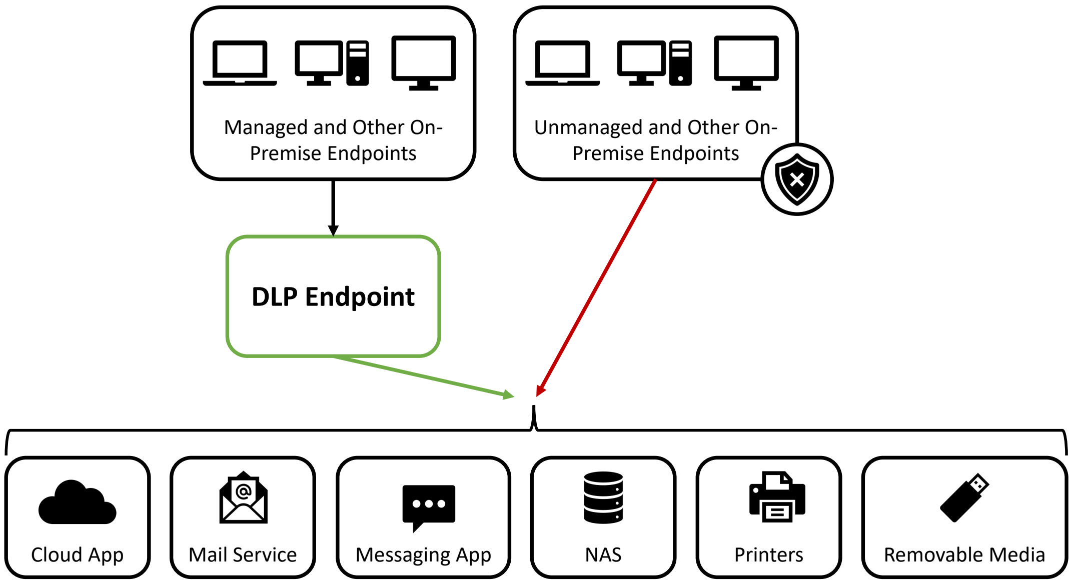

Managed and Other On-Premise Endpoints
Unmanaged and Other On-Premise Endpoints
DLP Endpoint
Cloud App
Mail Service
Messaging App
NAS
Printers
Removable Media

Figure 2.1: DLP System Protection

a condition for each field, making DLP rules more complex and less generic. Chapter 3 provides examples of such DLP rules.

Chapter 3

Case Study

The policy shown in Table 3.1 is used as an example throughout the entire thesis report. Each row in Table 3.1 corresponds to a rule. The way in which the conditions are stated is simplified to allow a concise rule presentation of the rule while maintaining a good understanding. The conditions of the first rule should be understood as follows: *If the operation is a file upload and the classification tag is secret then block the operation.* The conditions of the rule with priority 10340 can be read as follows: *If the operation is a file upload and the destination website is a company's website then the operation is allowed.* A full implementation of rule 2 can be found in the Appendix A. Each rule in this example will be referred as 'ruleX' in this report, where 'X' corresponds to the row in which the rule is located. For example, 'rule1' corresponds to the rule with priority 10310 and 'rule3' corresponds to the rule with priority 10330.

Table 3.1: DLP Policy Example

Priority	*Conditions*	*Action*
10310	fileUpload + secret	block
10320	fileUpload + confidential + to company's websites	continue
10330	fileUpload + secret + to company's websites	continue
10340	fileUpload + to company's websites	continue
10350	fileUpload	block
10360	fileDownload + from http site	block
10370	fileDownload + from https site	continue
10380	fileDownload + from github	block

This example illustrate key concepts of how DLP works and how administrators can create and manage rules. Priority numbers can be any integer and do not have to follow predefined values. 'fileUpload' and 'fileDownlad' refer to the type of system operation each rule will catch. Rule1's action will only be applid for uploads of files classified as secret. Secret and confidential correspond to the classification tag applied to a file. This policy aims to meet the following security requirement:

- file downloads are only allowed from https website
- file downloads from github are prohibited
- secret document uplaods are only allowed on a company's websites
- confidential document uplaods are allowed
- non-confidential document uploads are prohibited

These security requirements could be an example of what a company may want to enforce. The last security requirement is unlikely to actually exist in a company environment since, as a default behaviour, the company may want to allow file uplaods instead. This security requirement does not allow the uplaod of a public document, for example. This choice has been made only for the purpose of this example. Allowing secret file uploads on certain specific websites could, however, be a common use case. Allowed websites do not necessarily have to be company websites, but for the sake of this example, they have been chosen as the allowed option. Some may argue that company websites are in a known and secure environment, making uploading secret documents less not risky, which could justify the mentionned security requirement. Restricitions on the destination of file uplaods are also a common use case. More realistically, companies will have a list of non-allowed websites instead of a single one. The example rules provided in Table 3.1 do not meet the necessary security requirements. This issue will be discussed in the following paragraph

For this example, it is assumed that all files have already been classified. If this were not the case, a rule could be implemented to block file uploads of non-classified files. Such a rule could be written as follows: *fileUpload + classification tag is empty* with a block action. The positioning of this new rule illustrates well how priority numbers are responsible for enforcing certain security requirements. The security requirement in this case is that all uploaded files must have a classification tag. This new rule does not interact with the first three rules since they are triggered only on files that already have a classification. Placing this new rule after the rule5 would also not be a good idea since rule5 will catch all file uploads and the event will thus never reach the new rule. The last point to clarify is its position in relation to rule4. Rule4 allows file uploads only if the destination website is owned by the company. In this scenario, we must determine the priority between blocking non-classified uploads and allowing uploads to company websites. In other words, should we allow non-classified files to be uploaded to company's websites? Which of these two security requirements is more important? Let us assume that non classifed-files should not be uploaded regardless of their destination. In this situation, the new rule must be positioned before rule4. Otherwise, the new security requirement will not be enforced on company's website.

This DLP policy contains several anomalies, representing misconfiguations. For example, let's consider rule3, which is intended to detect file uploads of secret documents to a company's website and allow the operation. This rule is necessary in the context of another security requirement that specifies that no secret documents should be uploaded to the web. Rule3 works as an exception to the previously mentioned security requirement. Rule1 is reponsible for enforcing the fact that secret documents cannot be uploaded on the web. This reflects a misconfiguration, as in order for rule3 to be effective it needs to be placed before rule1. Therefore, its priority number is incorrect, and the security requirements are not being enforced due to this mistake. In the current situation, rule3 could be removed from the policy without changing the DLP behaviour. Changing the priority number of the rule and placing it before rule1 is a preferable solution to removing it. This way, it can work as an exception for the 'block all secret upload' requirement.

Another type of misconfiguration is illustrated with rule2, which allows the upload of confidential documents to company websites. However, rule4 states that, confidential file uplaods are allowed regardless of the destination website. In this situation, rule2 becomes can be removed as it is unnecessary. Another more generic rule enforces the same action for the event caught by rule2. The last misconfiguration present in this DLP policy is the rule8. Rule6 and rule7 together catch all file downloads since the source website is either HTTP or HTTPS. To prevent file downloads from the site https://github.com, rule8 should be placed before rule7. This way, it can act as an exception to the more general rule. The process of having a general rule with high priority numbers, and more specific rules with lower priority numbers, to better fit the security requirements is common in DLP administration.

An corrected version of the DLP policy can in found in Table 3.2.

Table 3.2: DLP Policy Example Corrected

Priority	*Conditions*	*Action*
10300	fileUpload + secret + to company's websites	continue
10310	fileUpload + secret	block
10340	fileUpload + to company's websites	continue
10350	fileUpload	block
10360	fileDownload + from http site	block
10365	fileDownload + from github	block
10370	fileDownload + from https site	continue

Chapter 4

Design

To aid administrators in managing DLP rules, the DLP analyzer must comprehend how the rules are constructed. This comprehension will enable the tool to identify unnecessary rules and any misconfiguration within DLP management. Eliminating unnecessary rules will result in a cleaner and more readable policy, which is crutial in preventing future misconfigurations. Due to the priority order, that the rules follow, some rules may interact with others and some of their events. THe tool should not only detect anomalies but also provide visualizations and summaries of DLP policies. Administrators can use the provided visualization and summary to identify anomalies and understand where misconfiguration occured. While some rule anomalies may not require suppression, misconfigured rules should be relocated within the DLP policy. Incorrect placement in the priority order can result in a rule being shadowed previous ones. However, this rule may still be necessary to enforce a security requirement, and therefore a more appropriate priority number should be assigned. The second goal of the analyzer is to suggest actions and priority numbers, when inserting a new rule into a policy. Inserting new rules is a common task in DLP administration, as the use cases they need to cover are not static and evolve over time. As loss vectors evolve, DLP rules must be updated by editing or adding new DLP rules. The case study (chapter 3) demonstrates that misplacing a rule can result in incorrect implementation of security requirements. To prevent such misconfigurations, the analyzer should, suggest a new priority that do not create anomalies or reduce the current security coverage.

DLP solutions can significantly affect endpoint performance. To mitigate this impact, the matching rule process should be as fast as possible. Ideally, more common rules should be placed at the top of policies to optimise performance. As priority numbers are crutial for the DLP's behavior, creating a new order should not alter the current security coverage. The analyzer's goal is to re-order rules based on hit probabilities. The last main goal of the analyzer is to provide a high-level overview of the collection data and exfiltration methods covered by the DLP rule set. This feedback will be expressed through a cybersecurity score, computed against risks mentioned in the MITRE framework. A high-level coverage of the DLP rule set will enable better coordination with other security software.

4.1 Anomalies

To create a tool that is universally applicable to any DLP solution, it is necessary to first define the structure of DLP. This generic representation should include definitions for rules, conditions, actions, events, and how rules can match events. The following concepts can be formalised as follows.

An *Action* authorises or blocks an event. The *Action* domain is defined as follows:

$$\mathscr{A} = \{block, continue\} \tag{4.1}$$

A *Field* is a property of an *Event*. The list of possible *Fields* depends on the DLP solution. The list of values that a *Field* can take is also dependent on the DLP. The domain of a *Field F* is denoted by $\mathscr{F}$.

An *Event* corresponds to a System Event, it can be formalised as a set of *Fields* and their values. The goal of the DLP rules is to catch some of these *Events*.

$$E = \{(F_0, f_0) \ldots, (F_m, f_m)\} \quad \text{where } f_k \subseteq \mathscr{F}_k \tag{4.2}$$

A *Condition* is a *Field* with a set of (valid) values for that *Field*. Equation 4.7 defines how an *Event* can match a *Condition*. Formally a *Condition* can be expressed as follow:

$$C = (F, v) \quad \text{where } v \subseteq \mathscr{F} \tag{4.3}$$

A *Rule* can be defined as a pair of a list of n *Condition*s and an *Action*. A *Rule* with priority i will be denoted as follows :

$$R_i = ([C_{i.0}, \ldots, C_{i.n}], A_i) \quad \text{where } A_i \in \mathscr{A} \text{ and } C_{i.k} = (F_{i.k}, v_{i.k}) \tag{4.4}$$

We define a function $R[F]$ which given a *Rule*, R, and a *Field*, F, returns the set of values of F that are valid to satisfy R. If the rule R has a condition on the field F, then the function returns the set of valid values of that condition, otherwise $\mathscr{F}$ is returned. If the rule R has no condition on F, this means that all values of that field can be caught by the rule, we return all possible values of F. Formally:

$$R[F] = \begin{cases} v_k & \text{if } \exists C_k \in R : F_k = F \\ \mathscr{F} & \text{otherwise} \end{cases} \tag{4.5}$$

In the same way, we define $E[F]$, which given an *Event*, E, and a *Field*, F, returns the set of values of E for that field, or all possible values of F. Formally:

$$E[F] = \begin{cases} f_k & \text{if } \exists F_k \in E : F_k = F \\ \mathscr{F} & \text{otherwise} \end{cases} \tag{4.6}$$

An *Event E* satisfies a rule R if for each condition of the rule, the intersection of its values and the values of the *Event* is not empty. If the intersection for a condition is empty, it means that the *Event* has a value that the rule does not catch, and so E does not satisfy R.

$$\forall C_{\in R} : v \cap E[F] \neq \emptyset \tag{4.7}$$

This formalised version of rules and conditions is the starting for building the analyzer. To define anomalies, we must first identify four different relationships that can exist between two DLP rules. Al-Shaer, Ehab and Hamed, Hazem introduced various rule relationships that can occur between firewall filtering rules [18]. These relations have been redefined in the section on DLP rules.

Two *Rule*s R_i and R_j are *disjoint*, denoted $R_i \nparallel R_j$, if they have at least one *Field* for which they have completely disjoint values. In other words, an *Event* will never match these two rules. Formally:

$$R_i \nparallel R_j \text{ if } \exists F : R_i[F] \cap R_j[F] = \emptyset \tag{4.8}$$

Two *Rule*s R_i and R_j are *equal*, denoted $R_i = R_j$, if each *Field* of the rules matches exactly. Formally:

$$R_i = R_j \text{ if } \forall F : R_i[F] = R_j[F] \tag{4.9}$$

R_i is a *subset* of R_j, denoted $R_i \subset R_j$, if all *Field* values of R_i are in R_j and R_i is not equal to R_j. Formally:

$$R_i \subset R_j \text{ if } \left(\forall F_{\in R_i} : (R_i[F] \subseteq R_j[F])\right) \land \neg(R_i = R_j) \tag{4.10}$$

Two *Rule*s R_i and R_j are *joint*, denoted $R_i \sim R_j$, if they are not *disjoint* nor *equal*, and none of them is *subset* of the other. Formally:

$$R_i \sim R_j \text{ if } \neg(R_i \nparallel R_j) \wedge \neg(R_i = R_j) \wedge \neg(R_i \subset R_j) \wedge \neg(R_j \subset R_i) \tag{4.11}$$

We define the operator ⊆ as

$$R_i \subseteq R_j \iff (R_i \subset R_j) \vee (R_i = R_j) \tag{4.12}$$

Table 4.1 shows rule relations for the five first rules in Table 3.1. Rule 1,2,3,4,5 are *disjoint* with Rule 6,7,8.

Priority	*10310*	*10320*	*10330*	*10340*	*10350*
10310	$=$	$\nparallel$	$R_3 \subset R_1$	$\sim$	$R_1 \subset R_5$
10320	.	$=$	$\nparallel$	$R_2 \subset R_4$	$R_2 \subset R_5$
10330	.	.	$=$	$R_3 \subset R_4$	$R_3 \subset R_5$
10340	.	.	.	$=$	$R_4 \subset R_5$
10350	.	.	.	.	$=$

Table 4.1: Rule Relations

Figure 4.1 visually represents rule relations. A circle represents all *Events* that a rule can match.

Assuming that a file cannot be both secret and confidential simultaneoulsy, rule1, catching all secret upload, and rule2, catching some confidential uplaods, will never match the same event as they are disjoint. However, rule3, which catches all secret uplaods to a company's website, is a subset of rule1 since every events that rule3 catches can also be caught by rule1. In fact, rule1 does not specify destination website and catches secret files regardless of their destination. Uploads to company websites can also be caught by rule1 and only their priority number will determine which rule is triggered. Finally rule1 which catches all secret upload event, and rule4 which catches upload to company's website are joint. There are events that can only be caught by rule1, events that can only be caught by rule4 and events that can be caught by both rules. For instance, uploading a secret file to example.com will only be caught by rule1. Uploading a public file to the company's website will be detected by rule4 but not rule1, as the file is public and not secret. However, uploading a secret file to one of the company's website can be caught by both rule1 and rule4, making them joint.

Based on relation defined before, we can define what is an DLP rule anomaly. A anomly should represent a misconfiguration or a situation where the purpose a rule is not applied well. A rule is called an *anomaly* if it can be removed from the policy without changing the policy behaviour not the cybersecurity coverage of the rule set. For example, a rule is an anomaly if no *Event* can match it because it is preceded by one or more other rules, that match all the events for which the rule was designed. Operations that could be matched by the *anomaly* rule are not a problem

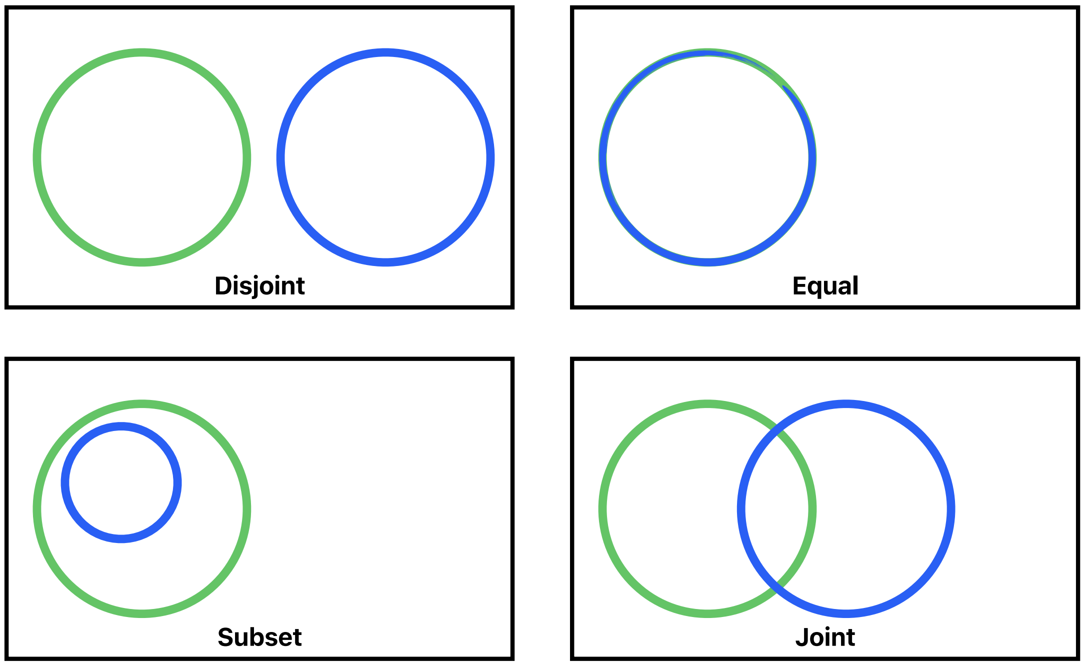


Figure 4.1: Graphical representation of rule relations

for DLP performance because they will be matched earlier. However, any other operations that reach the *anomaly* will be scanned unnecessarily, reducing performance. An *anomaly* can be removed from the rule set without changing its event coverage. However, an *anomaly* may reflect a misconfiguration, in which case, a better priority number should be found instead of removing the rule. Three types of anomalies are defined in this section. Golnabi, Al-Shaer, Voronkov, Katic, Benelbahri define several anomalies for filtering rules in firewalls, [6], [18], [21], [13], [2]. These anomalies have been adapted to DLP rules in the following section.

Shadow Anomaly: R_j is shadowed by R_i iff

$$(R_j \subseteq R_i) \wedge (i < j) \tag{4.13}$$

Set Shadow Anomaly: R_i is set shadowed by previous rules if the combination of several previous shadow R_i

Figure 4.2 illustrates the two shadow anomalies. A circle corresponds to the set of events a rule will catch. Circles in the foreground represent rules that have a lower priority (they will hit first). The anomaly rule is always the one shown in orange.

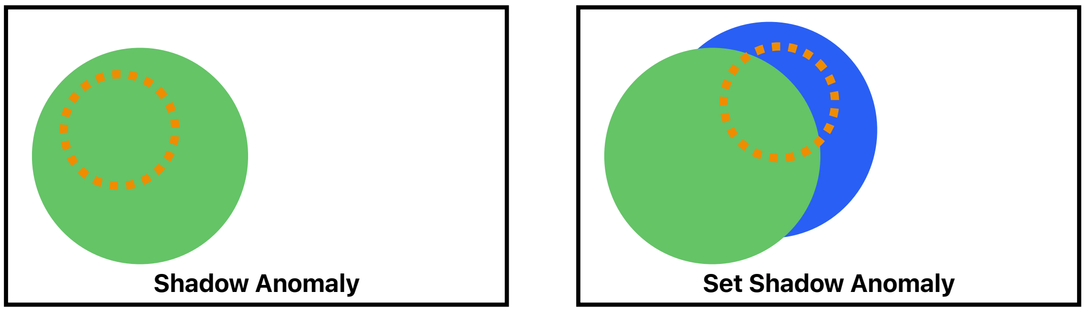


Figure 4.2: Graphical representation of Shadow anomalies

If the rule *fileUpload* + *secret* appears before the rule *fileUpload* + *secret* + *to company's website*, it will take precedence and capture all secret uploads, even if the destination website is a company websites, leading to a shadow anomaly. As a result, rule4 will never be triggered. An example of a set shadow anomaly is demonstrated with rule6, rule7 and rule8. Rule6 and rule7 together capture all file upload events, as one of them captures uplaods form HTTP websites and the other form HTTPS websites. Therefore, rule8 is set-shadowed and all events rule8 was intended for will captured beforehand. In fact, a file download form github necessarily originates from an HTTP or HTTPS source.

Redundancy Anomaly: R_i is redundant to R_j when R_i has the same action as R_j and its priority is lower and it is a subset of R_j. Before removing R_i we must also check that R_i is disjoint with all rules between R_i and R_j and have a different action. This way we can be sure that removing R_i will not change the behaviour of the policy. Formally:

$$(i < j) \wedge (A_i = A_j) \wedge (R_i \subseteq R_j) \wedge \left(\forall k_{\in\{i+1,\dots,j-1\}} : (R_i \not\parallel R_k) \vee \left(\neg(R_i \not\parallel R_k) \wedge (A_i = A_k)\right)\right) \tag{4.14}$$

To better understand the Redundancy anomaly, Figure 4.3 shows two different cases. For this anomaly, the action of each rule is important and is therefore written on the top right of each box. The left illustration shows a case where there is an anomaly, while the case on the right reprensents a use case where there is no redundancy because there is a *joint* rule between the orange rule and the green rule that have a different action. In the case on the right, the orange rule is therefore not an anomaly and should not be removed.

A final anomaly that can occur in DLP policies is when two rules have the same priority number. This may be prohibited by the DLP solution, but it mays also be allowed. In the case where two rules can have the same priority, it is the internal behaviour of the DLP solution that decides which rule is

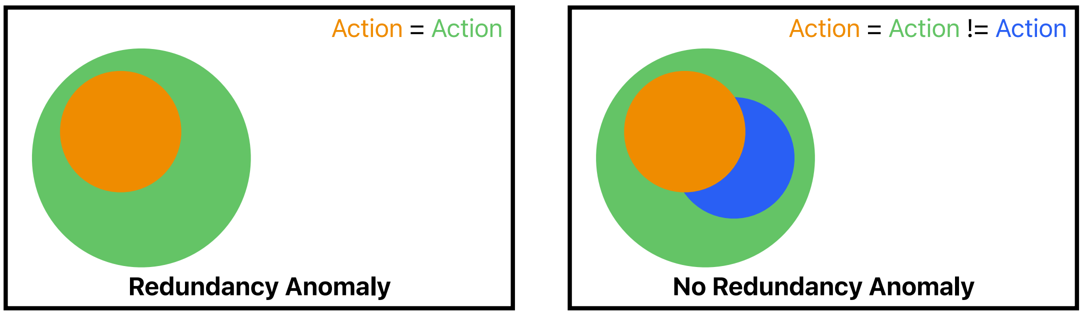


Figure 4.3: Graphical representation of Redundancy anomaly

evaluated first. Such a decision should not be based on technical details of the DLP agent. From the point of view of the analyzer, this is considered an anomaly, since it is possible to know which one will be evaluated first. This situation may highlight a configuration issue, and administrators may have overlooked this particular case. This anomaly is more common than what one could think. Several DLP solutions assign a default priority (hardcoded) to a new rule. If this default priority is not overwritten, we can end up with multiple rules with the same priority.

These design choices and definitions allow for the precise definition of abnormal DLP configurations. The general rule representation can also be used as a summary of DLP rules, which will improve rule readability for administrators. One trade-off of the formalization is that it simplifies DLP rules. Their action can be more specific than 'block' or 'continue', and they may be composed of variable definitions or perform actions of the file linked to the event, such a encryption, or removing a classification tag. These information are lost in the current translation. The trade-off is further discussed in chapter 5. To better assist administrators and comply with defined goals, it is also important to display warnings when the priority of two rules is the only factor responsible for the DLP behavior. This is not a misconfiguration, but rather a situation in which administrators must be careful, in order to correctly enforce secruity requirements. For example, rule7 captures all file downloads from HTTPS websites, and rule8, carptures all file download from github. These rules should be ordered carefully. As they are joint (a file download from https://something.github.com will can be cath by both rule) their order will be responsible for the DLP behavior. If allowing file downloads from https website is more important than blocking file download form github then the rule7 should have a lower priority number (it will hit first).

A rule *Warning* corresponds to a situation where an *Event* may match two rules. This is not necessarily an anomaly as one rule can represent an exception for a more general rule. In these situations, the priority number is really important, as it determines which of the two rules will actually match the *Event*. The order only matters if the two rules matching the same *Event* have different actions. Here are two different types of rule *warning*:

Overlap: Two correlated rules with different actions are considered an overlap

$$(R_i \sim R_j) \wedge (A_i \neq A_j) \tag{4.15}$$

Generalization: R_i is a generalization of R_j iff

$$(R_i \subset R_j) \wedge (i < j) \wedge (A_i \neq A_j) \tag{4.16}$$

For example in Table 3.1:

- Rule 4 *overlaps* with Rule 1
- Rule 5 is a *generalization* of Rule 4

Figure 4.4 graphically represents rule warnings.

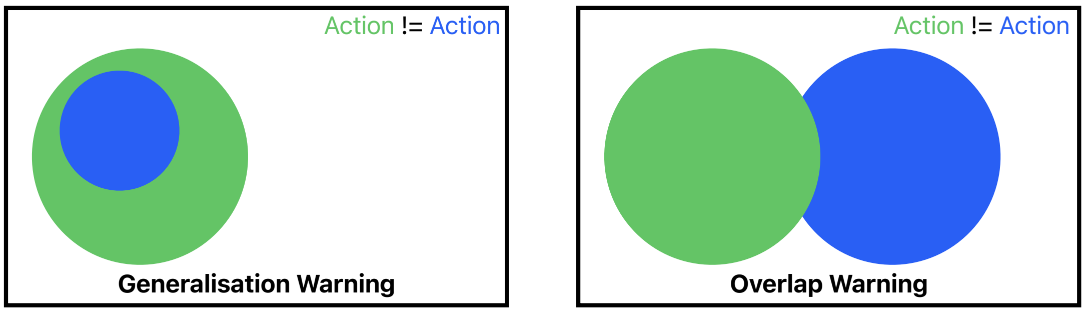


Figure 4.4: Graphical representation of rule anomalies

4.2 Policy Advisor

Current design description allows the analyzer to detect anomalies and issue warning for situation where only the priority number decides the DLP security coverage. A project goal was to help administrator extending the DLP data leak coverage. An important task in managing DLP software is to adding new rules to policies. This task is very common, as DLP policies are constantly changing, to meet the needs of the company. Inserting a new rule to a complex set of rule can be quite challenging, and can lead to serious errors. This section describes a new way of representing DLP policies: *Policy Trees*. This representation has two main advantages: firstly, it provides administrators with a good visualisation of their policies, and secondly, it can be used to help insert a new rule. A visualisation of a policy can be very important, as it allows the administrator to take a step back, and look at things from different angle. A trivial way of offering suggestions when inserting a new rule into a set of rules is to brute force. An algorithm could try to insert the new rule at every position and run the algorithm to detect anomalies. This method is neither not very efficient ($\mathcal{O}(n^3)$) nor very intuitive for the administrator.

The *Policy Tree* remains a representation of a policy, its components are then defined using the generalisation of DLP components.

- A *Node* represents a *Field*.
- An *Action Node* represents an *Action* and the priority of the corresponding rule.
- An *Edge* represents a *Condition*. It starts from a *Node* and goes to either another *Node* or an *Action Node*. The set of values of the condition is attached to the edge and it represents all values that satisfy the upper *node*/*field*.
- A *Path* is a list of *Edges*. The last node of a path should be an *action node*. A path represents a rule and we can expect the number of paths to be equal to the number of rules in the policy.

In this illustration we assume that each DLP rule has a condition based on the operation type, e.g., fileDownload, fileCopy, and so on. This is a reasonable assumption as rules are intended to block, or allow a specific event. A rule can still match multiple operation types. The policy tree is split into several parts, one of each operation type, to improve its readability. As this condition is assumed to be always present, it is not represented by an Edge or a Node.

An *Event* can follow several paths in a policy tree. To follow a path an event must satisfy every condition of its corresponding edge. When an event reaches multiple action nodes, we use the priority numbers to know which action will be enforced by the policy. If an event does not match any path, it is allowed by the policy by default (action = *continue*). For readability purposes, these default paths are not shown in the policy tree.

Figure 4.5 shows the generated policy tree for the example in Table 3.1, for the upload operation type.

Inserting a new rule into an existing policy is a challenging task, especially if the policy is large and complex. The new rule may be based on several operation types, and is likely to have interations with several existing rules. This section describes several concepts that are used to suggest action and priority for a new rule. Let's denote R_{new} the new rule that we want to insert into an existing policy.

First, existing rules that are *disjoint* with R_{new} will not generate a constraint on R_{new}. Two disjoint rules will by definition never match the same event, so the order in which they are evaluated

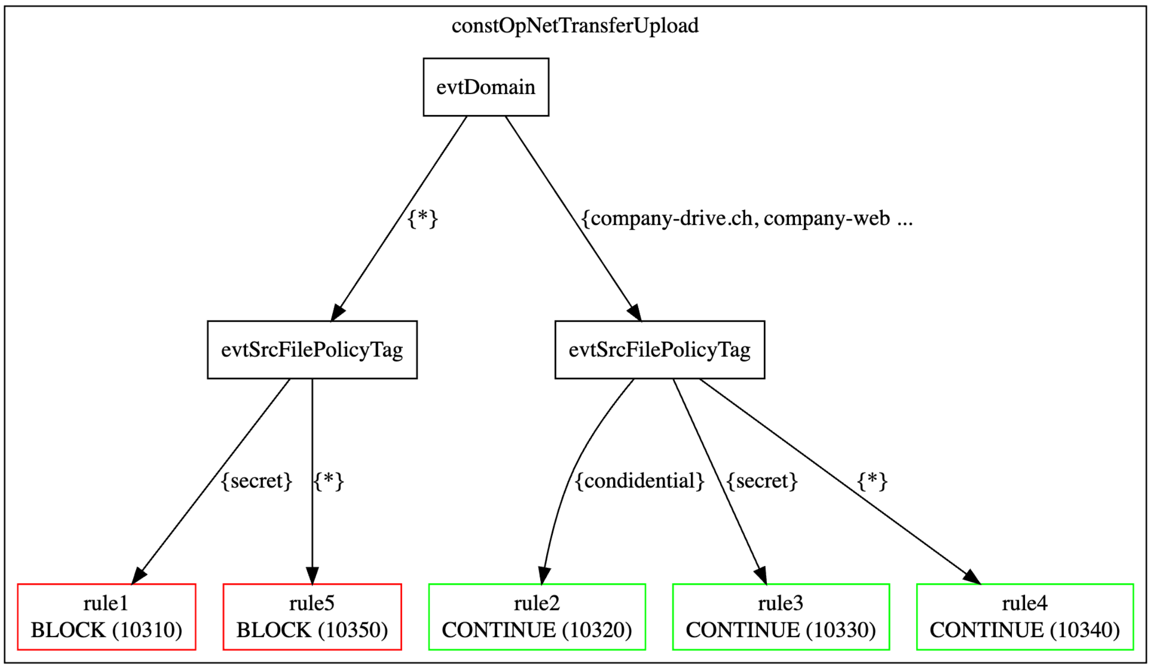


Figure 4.5: Upload Policy Tree

does not matter. R_{new} will likely have a non-empty intersection with several existing rules, in other work R_{new} will be *joint* with some rule R_i. In this case there is no risk of anomaly creation, but the position of the new rule and its action may 'shadow' part of an existing rule. This should trigger an *overlap warning*, see Equation 4.15, since the new rule might change the behaviour of the policy. The new rule may be a *subset* of an existing rule R_i. In this case, the priority of the new rule must be less than i. In this way, the new rule will be evaluated before R_i and will thus not be shadowed. Finally, in the situation where R_{new} is a *subset* of R_i, R_new will be placed before R_i, but its action should be different from that of R_i, otherwise it will be *redundant* with R_i. This last rule is only true if there is no *joint* rule between R_i and R_new, see Equation 4.14.

Rule for priority suggestion are summarized as follow:

- $R_{new} = R_i \implies$ do not insert the new rule
- $R_{new} \supset R_i \implies i <$ new priority
- $R_{new} \subset R_i \implies$ new priority $< i$
- $R_{new} \sim R_i \implies$ Warning
- $R_{new} \nparallel R_i \implies$ no priority constraint

Rule for action suggestion are summarized as follow:

- $R_{new} = R_i \implies$ do not insert the new rule
- $R_{new} \supset R_i \implies i <$ no action constraint
- $R_{new} \subset R_i \wedge$ no joint rule between R_{new} and R_i with action $\neq R_i$'s action $\implies$ new action $\neq R_i$'s action
- $R_{new} \sim R_i \implies$ no action constraint
- $R_{new} \nparallel R_i \implies$ no action constraint

A new rule can be constrained by more than one rule, all constraints must be satisfied and this allows us to simplify the final suggestion. For example, if a new rule is constrained by: priority < 100 and priority < 400; the final suggestion will simply be : priority < 100.

Note: Some suggestions may lead to impossible situations, e.g., $(x > 5) \wedge (x < 3)$.

A useless rule will be detected as an anomaly, but a rule that is misplaced can also lead to an anomaly. For example in Table 3.1, the rule 3 is detected as a shadow anomaly. Instead of removing this rule, we should try to reinsert the rule into the policy, to find a better, and correct, position. This will give us the following suggestion

Note: These suggestions have been direclty exported from the analyzer

- priority < 10310 or it will be shadowed by rule rule1 [BLOCK (10310)]
- action != BLOCK or it will be redundant with rule rule1 [BLOCK (10310)]
- priority < 10340 or it will be shadowed by rule rule4 [CONTINUE (10340)]
- if 10310 < priority < 10340 ==> action != CONTINUE or it will be redundant with rule rule4 [CONTINUE (10340)]
- priority < 10350 or it will be shadowed by rule rule5 [BLOCK (10350)]
- if 10340 < priority < 10350 ==> action != BLOCK or it will be redundant with rule rule5 [BLOCK (10350)]

They can be summarized as follow: '0 < priority < 10310 ==> action = [CONTINUE]'

In another context, this Policy Tree could also be used to match an Event with a Rule. This method could be faster than sequential checks, and evaluating the improvement would be an interesting further work.

The placement of the new rule could be motivated by a security requirement saying that no *secret* document should be uploaded to the internet. Table 4.2 shows the policy with this new rule.

Table 4.2: Policy with new a new rule

Priority	*Conditions*	*Action*
10310	fileUpload + secret	block
10320	fileUpload + secret or confidential + to github	continue
10330	fileUpload + to company's websites	continue
10340	fileUpload	block

In this policy, we know that rule 1 and rule 2 (the new rule) are *joint* (their intersection is not empty). This means that rule 2 has some unnecessary checks in its condition. In fact, rule 2 tries to match some operations that will always be intercepted by rule 1, due to the priority game. In the current example, this unnecessary condition is the *secret* check. All secret uploads will be caught first by rule 1. These unnecessary checks can have an impact on DLP performance and should therefore be optimised. Removing these unnecessary conditions can also improve the readability of the policy/rule. Having rules that perform unnecessary checks can only confuse DLP administrators. For performance and maintainability reasons these unnecessary checks should be removed.

To identify them we simply need to compute the intersection of two joint rules. This intersection corresponds to what should be removed from the higher priority rule. The intersection between rule 1 and rule 2 in Table 4.2 is *fileUpload + secret*. In this case, *secret* should be removed from rule 2. *fileUpload* should not be removed as this would lead to an empty condition.

It is important to note that this approach does not apply to *generalisation warnings*. In fact, removing unnecessary conditions from the higher priority rule would often require the creation of new rules. This is easily be understandable with rules that enforce a 'default behaviour', see rule 4 in Table 4.2 for a compelling example.

Priority suggestion, graphical representation of policies thanks to the policy tree and overlap simplification are direct implementation of the analyzer goal, that was to provide help in extendind DLP coverage. This solutions allow administrators to increase loss vector monitoring while not risking to insert new anomalies, that could reprensent a risk for the company.

4.3 Rule Ordering

Thanks, to the report's analysis of the DLP policy, it is clear that some rules are more likely to hit than others. To improve DLP performance, rules that are more likely to match should have lower priority (they are evaluated first). Since the matching rule is found earlier, fewer rules are evaluated and thereby increasing DLP performance. Re-ordering rules is not that trivial, as the coverage of the policy should remain unchanged. As we have seen, the rule order is very important and allows anomly avoidance. The challenge here is to find a rule order that preserves the policy intent while reducing the average time to match an event.

The same problem exists in the firewall world, Tapdiya, Katic, Hamed, ([20], [13], [11]) have already proposed solutions on how to tackle this problem. Tapdiya, [20], proposes a heuristic sorting technique to determine the order of rules in a firewall policy, which reduces the average number of rule comparisons while maintaining the intent of the policy. This technique provides good results and high performances. Coscia, [4], discussed the pros and cons of existing solutions.

The SGM method ([20]) applied to DLP policy would work as follows. A DAG, $G = (R, E)$ can be used to model a DLP policy, where R is the set of rules in the policy and E is the set of precedence relationships between rules. An edge exists between rule R_i and rule R_j if $i < j$ and the rules are not *disjoint*. The DAG corresponding to the example in Table 3.1 is shown in Figure 4.6

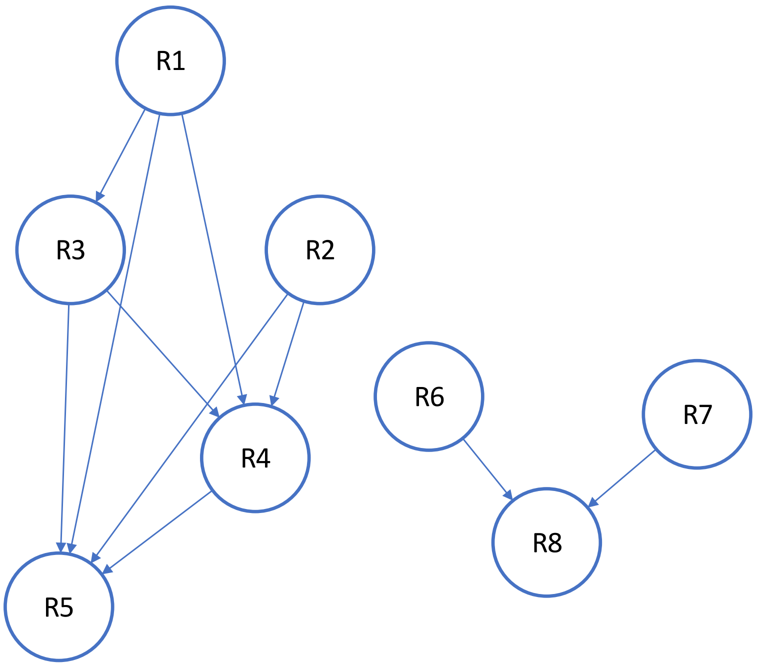


Figure 4.6: DAG corresponding to the example in Table 3.1

The problem of finding optimal firewall rule set ordering is a reduction of single job scheduling with precedence constraints [19]. Fulp, [5], has proven that the single job scheduling with precedence constraints is a $\mathcal{NP}-hard$, so finding the optimal DLP rule set ordering is also $\mathcal{NP}-hard$. A topological ordering of a directed graph is an ordering of its vertices in a sequence such that, for each edge, the starting vertex of the edge is earlier in the sequence than the ending vertex of the edge. This representation is not unique and is used by the SGM algorithm. Details on SGM algorithm are presented in [20], and will not be duplicated here. This algorithm is be implemented to perform the DLP rule reordering part of the project.

Note: This algorithm requires the probability that each rule has to hit. Company's log can allow us to determine the probability a rule has to hit. This data can be provided by the company.

4.4 Cyber Advisor

This section defines some general security requirements that a DLP should enforce. These security requirements are based on attack techniques defined in the MITRE ATT&CK [1]. Not all of the attacks mentioned in the framework are intended to be mitigated by a DLP. Based on these attacks, a coverage score is calculated and assigned to a given rule set. This score is followed by a visualisation showing further details. This information is intended to help security teams identify areas where they can improve their coverage.

Two types of tactics and techniques from the MITRE ATT&CK framework could be mitigated by a DLP solution. *Collection* techniques and *Exfiltration* techniques are relevant when computing a DLP coverage score. According to the framework, these types of data should be inspected since they pose a risk of data leakage.

- Archive Collected Data (T1560)
- Clipboard Data (T1115)
- Data from Cloud Storage (T1530)
- Data from Configuration and Information Repository (T1602)
- Data from Local System (T1005)
- Data from Network Shared Drive (T1039)
- Data from Removable Media (T1025)
- Data Staged (T1074)
- Email Collection (T1114)

- Screen Capture (T1113)
- Video Capture (T1125)
- Audio Capture (T1123)

These techniques should also be taken into consideration for complete DLP coverage

- Exfiltration Over Alternative Protocol (T1048)
- Exfiltration Over Other Network Medium (T1011)
- Exfiltration Over Physical Medium (T1052)
- Exfiltration Over Web Service (T1567)
- Transfer Data to Cloud Account (T1537)

Determining whether a rule set covers these use cases is highly dependent on the DLP solutions and vendors. Each DLP may have different ways of writing conditions, so it is not possible to create a generic way of building this score.

To verify that a rule set covers these use cases, we translate each requirement into one or more events. The mapping between security requirements and events will be DLP dependent and cannot be generalised. Once these events are created, the Policy Tree described in section 4.2 is used to match them them to one or more rules. If no rules match a created Event, then the security requirement is considered as not covered by DLP policies.

This feedback implements the analyzer's goal of providing high-level feedback on DLP security coverage. This part remains specific to DLP solution but can be easily adapted using a tool that would translates high-level requirements into DLP events. Any non-covered data types or exfiltration methods should raise awareness in the company environment to ensure that future deployments or other software take these risk into account.

Chapter 5

Implementation

All the design research done in the previous chapters is generic to any DLP solution. During the implementation, the first challenge was to adapt all this theory to a real world configuration. The analyzer should remain as generic as possible and be easily adaptable to other DLP solutions. The implementation was therefore split into two main distinct parts. The first one is a *parser* that converts all the company's rule into a generic format. The normalization step is necessary to maintain high adaptability of the analyzer. The generically formatted rules are then used as input to the second part of the analyzer. The second part of the analyzer, will perform all the core calculations and optimisations. This way, the core of the program is easily reusable with different solutions. The *parser* acts as a connector between a proprietary solution and the analyzer logic. A different parser is required for each DLP solution that should be supported. In the scope of this project, only one parser was implemented, to parse corporate policies.

For a better integration and ease of use, the parser is built to process the export file from the DLP administrator console. Polices and rules are configured using a GUI from the DLP administrator console. The management console has an option to export all rules to an XML format. This export includes, among other thing, the rule name, the priority, the conditions and the action. Conditions used by multiple rules can be written to a *Component Rule*. Component rules can later be referenced in classic rules. The Management Console also allows the administrator to configure *Component Lists*. These lists can be used to define, for example, sets of file extensions or sets of websites, used by multiple rules. The parser can also parse and integrate component rules and component lists. Some part of DLP rules do not evaluate to true or false, they are then skipped by the analyzer. For example, a rule may set a variable that is later printed in the log file. Such statements are not considered conditions, and the analyzer will dismiss them. Finally, in order to compute a new rule order, the parser must also handle a CSV file with rule hit probabilities.

The analyzer was implemented in Java. This choice was motivated by two main points: it is an object-oriented programming language and it is already well integrated into the company's environment. Object-oriented programming was very convenient because it allows the analyser to follow closely the theoretical part, e.g., Rule, Condition, Action, Node, etc, are each represented by a class. The DLP analyzer, is a tool that will be run only when changes are made to DLP policies. As such, the analyzer does not need to have very high performance and programming languages such as C++ were thus not required.

The implementation revealed some challenges that were not covered in the theoretical part. First, in practice, a rule may contain OR statements. OR statements contain several conditions, and only one of them has to evaluate to true, to trigger the action rule. In Equation 4.4, a rule is defined as a list of conditions, and all of which must evaluate to true to trigger the rule action. The definition assumes that the conditions in a rule are in an AND statement. To be consistent with the theory, the parser will split a rule with an OR statement into several rules, each made with only one AND statement. For example, a rule with conditions : $A \wedge (B \vee C)$, where A, B, C are conditions, will result in rule $A \wedge B$ and rule $A \wedge C$. The complexity of a rule increase because an OR statement can contain AND statements, or even other OR statements, and so on. A recursive To recursive algorithm is used to parse a rule, since the 'depth' of the rule is not known in advance. A rule with many nested ORs and ANDs can cause the analyzer to run out of memory. This case is known, but DLP rules also need to be parsed by the management console, and they should make sense to a human administrator, this way we assume that their complexity will not be such that the analyzser runs out of memory.

To satisfy a condition, Equation 4.7, the event should consist of at least one element from the condition values. In practice, some rule conditions can enforce multiple values for a given field. For example, a rule can be written as follows : *fileUpload* + *tag is A and B*, where A and B are different classification tags. This situation does not correspond to the definition described in chapter 4. To be consistent with the theory, the parser splits such conditions into multiple one. A rule can then have several conditions for the same field, this must be taken into account by the analyzer. The rule, *fileUpload* + *tag is A and B*, will be rewritten as : *fileUpload* + *tag is A* + *tag is B*.

Two rules are considered *disjoint* if no event can be caught by both of them, in other words if the intersection of one of their conditions is empty, see Figure 4.1. For example, rule : *fileUpload* + *tag is A*; and rule : *fileUpload* + *tag is B*; are disjoint because the intersection of their tag conditions is empty ($\{A\} \cap \{B\} = \emptyset$). This conclusion is correct in theory but the implementation phase revealed that this is not always true. In fact, there are some fields (such as classification tag) for which an event can contain several values. The file link with an event can actually be classified as A and B. In this way, the two rules presented above can both apply to a same event and should not be considered as *disjoint*. These fields are special cases where the theory does not apply. They should be hardcoded into the analyzer, and the intersection of two such conditions should never be considered empty. These fields are DLP solution dependent and cannot be generic to all DLP solutions.

Keeping the core of the analyzer generic for any DLP solution, has resulted in some constraints. The set of valid values of certain fields depends on the DLP solution and cannot be generalized, also it is difficult to define a finite set of valid values for certain fields, such as destination file path.

This limitation makes it harder to evaluate certain conditions. For instance a condition specifying that the *file tag is not secret,* should be tranlated as *file tag is in {all tags} \ {secret}.* The *{all tags}* part of the condition, cannot be defined in a generic environment, and the same applies to certain fields such as file paths and source web URLs. Without a proper definition for the set of possible values for each field, assumptions and simplifications were made when computing the relation between two rules. For example, when trying to compute the intersection of two conditions, such as condition A (*tag not in {secret}*) and condition B (*tag not in {confidential}*). To compute the intersection of A and B, we need the list of all possible tags. If the list only includes *secret* and *confidential* then $A \cap B = \neg\{secret\} \cap \neg\{confidential\} = \{confidential\} \cap \{secret\} = \emptyset$. However, if there are other possible tags (e.g. *{secret, confidential, public}*), the intersection of A and B will not be empty ($A \cap B = \{public\}$). To address this issue, we can either assume that the intersection is not empty in this scenario which may result in non-existent rule relations. Assuming more rule relations than what actually exist is preferable to assuming fewer, as fewer relations may result in missing anomalies. Alternatively, this part could be coded as dependant on the DLP solution, resulting in more precise rule relations but sacrificing the generality of the analyzer.

As mentioned in previous chapters, DLP rules can have several actions. These actions have been summarised as *block* and *continue*. However, there are certain rules that cannot be associated with either of these two conditions. The *block* and *continue* actions assume that subsequent rules will not be evaluated, as an action is performed on the system operation. In practice some rules are defined as 'continue rule after evaluation'. This option stands for rules that do not perform an action on the system operations. The evaluation of the rule set will not stop if one of these rules matches the event. Since 'continue rule after evaluation' rules do not have an action, in the sense of Equation 4.1, they cannot create anomalies nor warnings. These rules may set variables, or simply log certain system operations. They are not meant to enforce specific security requirements. These rules are ignored by the analyzer.

By default, the analyzer parses all given rules and displays a summarised and English version of each of them. Anomaly detection is then performed, and any detected anomalies are fed into the insertion algorithm, to suggest a better priority assignment. In addition default behaviour, several options can be enabled to enhance the analyzer behaviour. These options correspond to features of the analyzer that are not used regularly. The first available option is to enable all warnings. Warnings are disabled by default because DLP policies often have many *joint* rules, and the output can become very noisy. Warnings are usually already known to administrators and voluntary. Another option is to enable the graphical generation of the policy tree. This option generates a Graphviz ([8]) input file representing the policy tree. The –new-rule option allows the user to ask the analyzer for a priority and action suggetion for one or more new rules. A final option is used to generate a new rule order based on given rule probabilities.

The output of the analyzer is in text format, requiring the user or administrator to go through the result and select what is needed for the current situation. This is an acceptable solution for features that will not to be used regularly. However, anomaly detection should not require more work from the DLP administrator and it should not add complexity to the company's workflow. The team managing the DLP is currently using git to review and manage their rules. Branches are created for each bug fix, feature and pull request are used before any merge into production. The analyzer has also been integrated into this workflow and is run automatically when a pull request is created or modified. A 'build status' is then generated based on the lastest commit changes. If any of the last commit changes lead to an anomaly the pull request will show a the 'failure' status. This way the analyzer is seemlessly integrated into the team's existing workflow. This integration required some code refactoring, external scripts and configuration.

Chapter 6

Performance Evaluation

This section describes experiments that have been tried in order to measure the performance impact that a rule in a policy can have. In other words, we want to quantify the actual execution time difference relative to the position (priority number) of the first matching rule. The DLP evaluates each rule in the rule set sequentially until it finds a matching rule. Theoretically, the process of finding the matching rule should take longer if the matching rule is position 50, than if it is in position 3, for example. Quantifying this difference would allow the improvement offered by a new rule order to be assessed as well as the actual performance benefit of removing anomalies.

6.1 Experimental Setup

All experiments were run on a dedicated machine. The first reason for this choice is technical. In fact, it is not possible to properly install the DLP solution on a machine that is not part of the company's network. Secondly, the results obtained are more reprensentative of the company's environment. A disadvantage of such a choice remains the fact that the company's computers also have several software programs installed, such as an Endpoint Detection and Response (EDR), which can create noise during the testing phase. For security reasons, these other agents could not be disabled. To filter out potential noise, the experiement was run several times. For technical reasons, the test machine runs on MacOS, not on Windows, and is a physical machine. The test machines are a Mac Studio with an ARM-based System-On-Chip Apple M1 Max running MacOS 13.4.1 with 32GB of memory, and a MacBook Pro with an ARM-based System-On-Chip Apple M2 pro running MacOS14.2.1, with 16GB of memory. These computers will not be connected to the internet and no software other than the Terminal, will be manually started. The experiment will only use the local file system of the machine and not any Network Area Storage. System operations with network area storage or network operation are not used for the experients as they are too dependent on network speed, which can vary greatly.

10 000 files have been created using the following script Listing 6.1. These files are used for each experiment without being regenerated. They are stored in 'folder1'.

Listing 6.1: File creation script

```
#!/bin/bash

# Content to write to each file
content="non-empty␣content"

# Use a loop to create the files
for i in {1..10000}; do
    echo "$content" > "dlp_experiment_pattern_${i}.txt"
done
```

6.2 Experiement Design

The first experiment consists of two runs. In the first run, the DLP is enabled and all rules are deployed plus the Rule in Table 6.1 in the first position (= lowest priority number). For the second run, the DLP will be enabled and all rules will be deployed plus the rule in Table 6.1 in last position (=highest priority number). This rule will match the file move operation of every 10,000 files previously created (see section 6.1). None of the other deployed DLP rules will catch events generated by the experiment. The experiment will consist of moving the 10,000 files from 'folder1' to 'folder2' and then moving back to 'folder1'. File are moved one at the time. These steps are performed one hundred times. Script Listing 6.2 shows exact operations performed during the experiment. Time measurements are taken after every 10,000 moves. An experiment run will perform 2 000 000 file moves ($10000 * 2 * 100$)). An experiment run will thus provide 200 time measurements. We expect to see a small difference in time measurements depending on the placement of the matching rule. File moves should be faster when the rule in Table 6.1 is in first position than when it is last position. This experiment is performed on the Mac Studio, which is rebooted before each run.

The second experiment will also run the 2,000,000 file moves, but will update the priority of the matching rule every 20 000 file moves (one round trip). The priority is updated so that it matching rule moves from the first to the last position. A priority update is usually perform thanks a communication with the DLP administrator console. However, in this experimentation, the test machine is not be connected to a network. This communication is therefore impossible. To get around this, the DLP agent's security protection was disabled. This allows us to directly edit rules direclty on the local machine. This way, the priority of the rule was updated without any network communication. The agent must be restarted for the change to take effect. In this experiment we ensure that the test machine is in the same state while the rule has a different priority. When the testing machine is restarted, it is difficult to ensure that its behaviour will be identical to the previous run.

Table 6.1: Experiment Rule

Priority	*Conditions*	*Action*
x	fileMove + name contains "dlp_experiment_pattern"	continue

Listing 6.2: Experiment script

```
#!/bin/bash

# Define the source and target folders
SOURCE_FOLDER="folder1"
TARGET_FOLDER="folder2"


for round in {1..100}; do

  # Use 'gdate' to get the nanoseconds
  start_time=$(gdate +%s%N)

  # Move each file with pattern 'dlp_experiment_pattern_NUMBER.txt'
  for i in {1..10000}; do
    mv "${SOURCE_FOLDER}/dlp_experiment_pattern_${i}.txt" "${TARGET_FOLDER}/"
  done

  end_time=$(gdate +%s%N)

  # Calculate the time taken to move the files in nanoseconds
  time_taken=$((end_time - start_time))

  start_time=$(gdate +%s%N)

  # Move each file that matches the pattern 'dlp_experiment_pattern_NUMBER.txt'
  for i in {1..10000}; do
    mv "${TARGET_FOLDER}/dlp_experiment_pattern_${i}.txt" "${SOURCE_FOLDER}/"
  done

  end_time=$(gdate +%s%N)

  time_taken2=$((end_time - start_time))
```

```
    echo "${round},␣${time_taken},␣${time_taken2}"
done
```

6.3 Results

Figure 6.1 shows the result of the first experiment. Each cross corresponds to the time needed to perform 20,000 file moves. The blue crosses correspond to times when the matching rule was in the first position, and the red crosses correspond to times when the matching rule was in the last position. The two lines of the graph correspond to the average time of each run of the experiment. On this graph, the average time when the rule is in the last position is lower than when the rule is in the first position. This is the opposite of the expected result. The experiement was run 3 times and there was no real time difference between each run of the experiment. Time measurements were taken in nanoseconds and no significant difference could be detected between the two runs of the experiment. This result is also confirm by the Figure 6.2. This graph represents times needed to move 20,000 files while toogling the priority of the matching rule. The red crosses corresponds to the times when the matching rule was in the last position. In this experiment, the average time when the rule is in the last position is again lower than when it is in the first position. There is a significant time difference between the two experiments, and this is simply because the experiments were not run on the same physical machine.

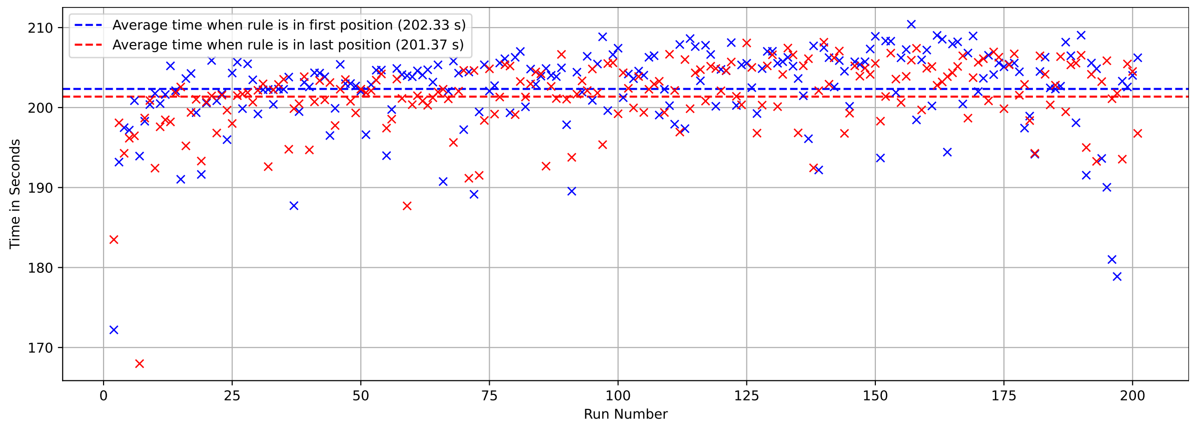


Figure 6.1: Time to move 20,000 files on mac Studio, run after reboot

Theoretically, and based on how the DLP finds the matching rule, we should have found a small time difference between each run of the experiment. The experiments showed that this was not the case. It may be that the experiemental setup is too noisy to extract the theoretical time difference. However, in the context of the company, it is not possible to have a less noisy environment. Another explanation could simply be that the time difference when the rule is in first or last position is

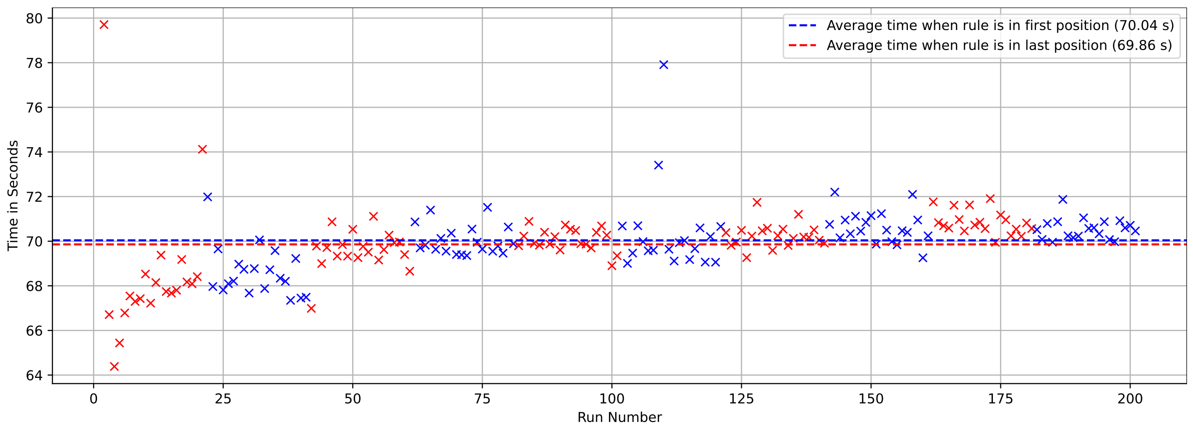


Figure 6.2: Time to move 20,000 files on Macbook with priority toogle

simply not significant. The time overhead that the DLP imposes to a system operation is simply too high (double the time when DLP is off) and makes the position of rule insignificant in terms of performance. One way to improve our experiment could be to add a few more rules to the environment, so that their impact is theoretically greater. For the current experiment, 35 DLP rules were deployed on the test machine. Due to company's constraints, it was not possible to deploy more rules on the machine. Lastly these result may suggest that the matching rule process is not sequential, even if the DLP solutions say it is. The sequential matching may simply an easy high level describing of the DLP behavior. In fact administrators might not need to know technical details in order to write and manage DLP rules. Further experiements could try to precisely understand the matching. Such would allow reasearch on how to improve the matching process, this is however outside the scope of this project.

Chapter 7

Discussion

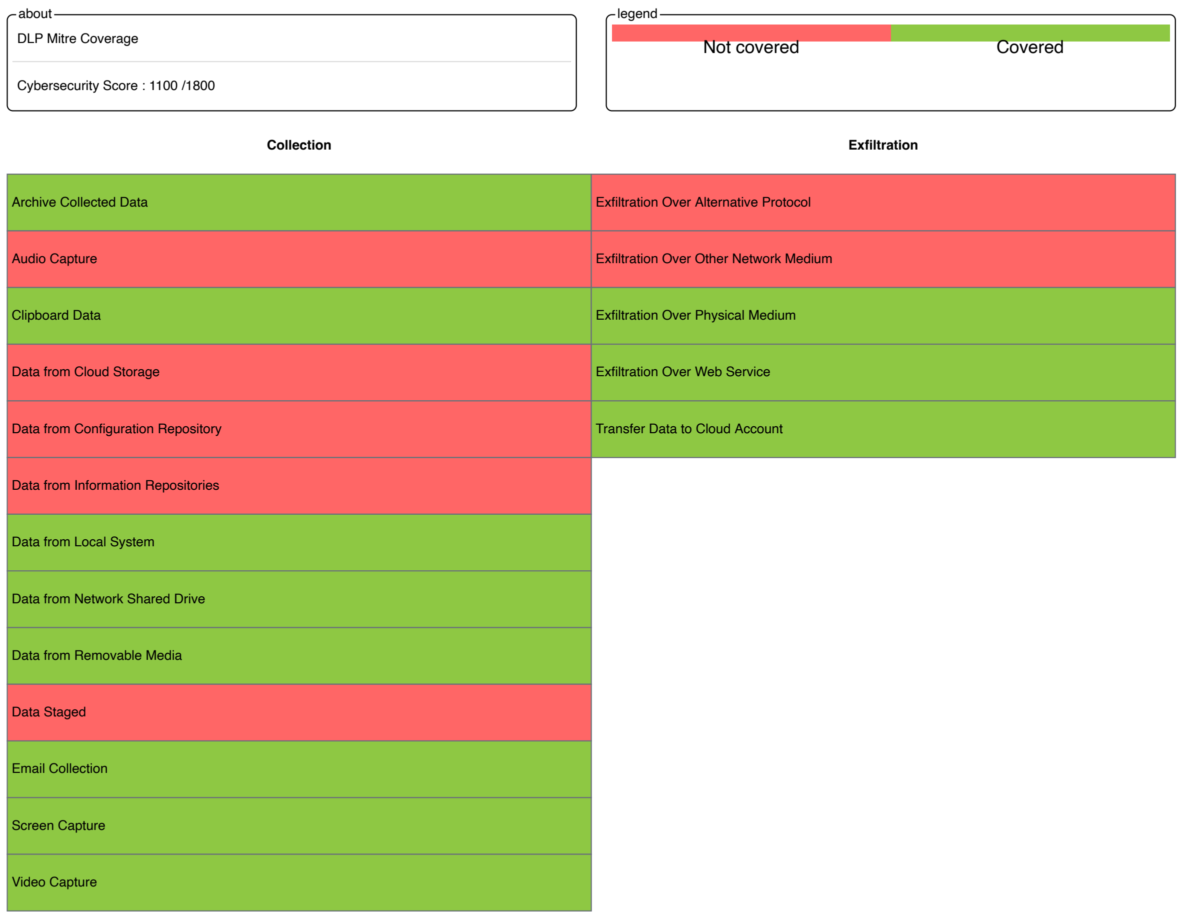
How the analyzer has helped the data security teams is hard to quantify. The analyzer has already been adopted by the team. On the one hand, it is very useful, as an accessible documentation of each rule. In fact, the parsed version of the rule is more readable than the XML version written in the management console. Also, in the administrator console, not all the information about a rule is in the same place, the summary provided by the analyzer gathers all the useful information, for all the rules in one place. This parsing output is now the first and main source when discussing rules enhancements. It is also the starting point for any new feature or bugfix. Improving the readability of DLP policies is an important step in reducing misconfiguration. On the other hand, the visual representation of DLP rules via the policy tree, is not convincing. The resulting policy tree is very large and cannot fit on a screen, which requiring the user to scroll and manually (or via ctrl+f) search for some information. The policy tree shows rules after they have been split by the parser, and an event can follow several paths (several rules). This makes the policy tree hard to digest, and is not really used in practice. However, it can remain a nice representation for small rule sets, as shown in Figure 4.5.

The anomaly detection algorithm was run on company's rules that are currently in production. The analyzer found several different anomalies. With the help of DLP administrators, these anomalies were theoretically confirmed as such. Some rules were sharing the same priority number. This type of anomaly can be difficult to catch as priorities are managed on a rule-by-rule basis. Rules with the same priority number were disjoint, so it did not reprensent a leakage risk. Some rules were flagged as a shadow anomaly. In this case, they did not represent a risk of data leakage, but could have had a negative impact on performance for some events. They were not missplaced, but simply surplus to requirements. The shadow rules were splitted versions of other rules containing OR statements. Rules with several ORs were necessary, but some conditions in OR statements, were unnecessary, so resulting splitted rules were already covered by the DLP policy. The last anomalies were redundant anomalies. Several redundant rules were already known to the teams and were actually there on purpose. The DLP solution used by the company allows fine control over which rule can log what. These redundant rules were created to catch a specific event and simply log the corresponding event. The more general rules responsible for the redundant anomalies, catches a lot of events, and it is not appropriate to let them log all of them. In this situation, the creation of redundant rules was necessary and allows finer control over the events to be logged. The analyzer does not take such settings into account, so these anomalies were ignored.

The algorithm that suggests priorities when inserting a new rule into an existing rule set, was used by the team even before it was integrated into the team's git workflow. As this feature only provides suggestions, DLP administrators use it to confirm the original priority they thought. This acts as a double check, giving more confidence when adding a new rule. This feature is particularly valuable when playing with rules that have a significant impact on DLP policies.

The cybersecurity score provided by the analyzer is used in two different areas. The first is within the data security team, where this score is used as coverage rate for the Data Leakage tactics mentioned in the MITRE framework. From a DLP perspective this score should not decrease as it would mean that DLP coverage is reduced. The score has also been included into reports, presentations and dashbords associated with the DLP. The second areas makes more use of the list of exfiltration methods and collection data not covered by the DLP. The company uses this list as a requirement for other software, or future projects. Anything not covered by the DLP will be covered by another tool/agent. For example, the DLP does not protect against data exfiltration over various network protocols, such as telnet. This use case must therefore needs to be covered by another tool, in this the firewall can block telnet traffic. The cybersecurity score is accompagned with visualisation of the DLP coverage. En example of such visualisation can be found in Figure 7.1.

The process of generalizing rule representation was necessary to build an analyzer that would remain generic to any DLP solution. This way, only the translation between a proprietary format to the generic rule representation on required to adapt the analyzer to another DLP solution. This design choise had some drawback since it also simplified the actual capabilities of a DLP rule. This have further discussed in chapter 5. This trade-off between complexity and usability has been an ongoing challenge during the development phase. Allowing more precise anomaly detection would require non generic assumption. As a first DLP analyzer the choice have been made to remain a generic possible, event at the cost of less precise suggestions. For instance not knowing the entire set of available classificatin tag implies more difficulty in detected set shadow anomalies. For example knowing that there exist only the tags: *secret*, *restricted* and *public* would allow the analyzer to find more precise anomaly. An unrealistic but convincing is the following rule *fileUplaod + secret or restricted or public*, any fileUplaod which priority is higher will be shadowed. This conclusion is only possible if the analyzer become specific the a DLP solution. One could think that some precision could also have been traded off for better performances. Howerver since the analyzer is not meant to be run regularly, it is not key that its computation are really efficient. During the implementation, effort where made to chose algorithm, data structure and conception choice that efficient, but not at the cost of precision.


about

DLP Mitre Coverage

Cybersecurity Score : 1100 /1800

legend

🟥 Not covered 🟩 Covered

Collection	Exfiltration
🟩 Archive Collected Data	🟥 Exfiltration Over Alternative Protocol
🟥 Audio Capture	🟥 Exfiltration Over Other Network Medium
🟩 Clipboard Data	🟩 Exfiltration Over Physical Medium
🟥 Data from Cloud Storage	🟩 Exfiltration Over Web Service
🟥 Data from Configuration Repository	🟩 Transfer Data to Cloud Account
🟥 Data from Information Repositories	
🟩 Data from Local System	
🟩 Data from Network Shared Drive	
🟩 Data from Removable Media	
🟥 Data Staged	
🟩 Email Collection	
🟩 Screen Capture	
🟩 Video Capture	

Figure 7.1: DLP Mitre Coverage

DLP is not an active reasearch topic, and the primary actors involved in its evolution and its improvement are the DLP companies themselves. There are several DLP solutions available on the market, and it is hoped that this competition will only improve DLP capabilities. As discussed earlier, a parallel can be drawn between firewalls and DLPs. Gouda and Liu proposed a method called structure firewall design to reduce redundancy in the firewall policies [7]. To enhance the firewall's performances, Katic and Pale proposed a solution for rule optimization [13]. To reduce the size of firewall rule set, two algorithms, i.e., SSO (simple substitutional optimization) and CSO (complex substitution optimization) have been proposed by Yoon, Chen and Zhang [22]. Additionally, Tapdiya and Fulp proposed a heuristic sorting technique called sub graph merging (SGM) algorithm [20]. The algorithm improves the firewall performance by reducing the number of rule comparisons required per packet and re-order the rule set. In addition, it moves the most frequently matched rules on the top of the rule set. The algorithm, however, does not have the capability to discover anomalies, i.e., redundancy, correlation etc. in the rule set. The effectiveness of these techniques has been discussed by Khan and Bilal [14]. Katic and Tihomir [13], Voronkov and Artem [21], Chomsiri and Thawatchai [3], and Al-Shaer and Ehab [18] all define and describe anomalies in firewall rules and provide solutions for detecting them. The recent work of Hakani and Dhwani [10] analyses these various types of rule set anomalies ii firewall systems and present different trade-offs of each solutions. While research on firewalls is extensive, adapting it to DLP solutions may improve current solutions. The lack of reasearch papers on DLP solutions and challenges is understandable, as it has not achieved the same level of success has firewalls.

Due to time limitations and the need to remain generic, the analyzer cannot detect set-shadow anomalies. Further work could focus on implementing this feature. The brute force technique cannot be applied as it would not be polynomial time. To check whether a rule with priority i is set-shadow, one could construct a 'union rule' of all rules that have a priority lower than i. Then, the simple anomaly detection algorithm would determine if rule R_i is set-shadowed by previous rules. However, this will not reveal which previous rules are responsible for such the anomaly. Detection and definition of set-redundancy in a DLP rule set would also improve the anomaly coverage of the analyzer. An important next step in improving the analyzer would be to build a more complex and generic rule representation. This representation could handle OR statements, variable assignments, and rule that perform actions on events (such as encryption, removing classification tag). It could also manage rules that do not stop the matching process. Future work could also focus on improving corporate integration, by retrieving rules directly from the administrator console or database. The simplification of overlaps can also be used to simplify existing rules, making priority numbers even more responsible for security requirement coverage. To improve performance measurements of DLP rule order, it may be beneficial to estabish a better environment and conduct more precise experimentation. However, if performance is the primary objective, research should focus on how to optimise the DLP itself rather than the rule set. For instance, the DLP integration into the agent's system could be drastically improved by finding an efficient matching rule algorithm, rather than a sequential one. Automatically creating DLP process exclusion or file exclusion is likely to have a measurable performance impact on the global user work environment. The capabilities of DLP solutions are still expending and reasearch on how to enhance DLP performance, or how to optimize rule sets may lead to better features being implemented directly into the product itself

Chapter 8

Conclusion

In summary, the principal outcome of this project is the development and implementation of a DLP analyzer, specifically tailored to enhance the managment of Data Loss Prevention (DLP) policies. This tool does not directly accelerate DLP performance; instead, it significantly aids administrators by ensuring that the policies are coherent, free from anomalies, and effectively prevent data leakage, which is the foremost function of DLP systems. By automating the detection of redundancies and conflicts within the rule set, the analyzer enables administrators to amintain clean and concise DLP policies, which is crutial for safeguarding against data loss.

The core results indicated that while direct improvement in DLP system performance is limited, the indirect benefits of utilizing the analyzer are substantial. The analyzer simplifies the complexity of rule management, alerts administrators to overlaps and potential rule shadowing, and ensures the integrity of data security requirements. In essence, it works behind the scene to provide a structured and dependable framework that underpins the efficacy of DLP systems. The reliability and clarity it brings to policy administration means organizations can trust their DLP's operational directives and reduce de risk of unintended data exposure.

By focusing on the optimization of rule-based management, the analyzer fortifies the preventive aspect of data protection strategies. It addresses the necessity of maintaning unambiguous policy rules, thereby sustantially decreasing the likelyhood of security breaches due to policy errors. This is particularly important for complex organization where DLP administration is a continuous challenge. Althought performance enhancements are best realized through precise process exclusions, the clarity and precision provided by the analyzer deliver a stronger, more resilient framework for DLP systems, ensuring that the overarching goal of data security is robuslty and efficiently met.

Bibliography

[1] MITRE ATT&CK. *MITRE ATT&CK.* 2023. URL: `https://attack.mitre.org/`. (accessed: 2023.10.10).

[2] Mohammed Anis Benelbahri and Adel Bouhoula. "Tuple Based Approach for Anomalies Detection within Firewall Filtering Rules". In: *2007 12th IEEE Symposium on Computers and Communications.* 2007, pp. 63–70. DOI: `10.1109/ISCC.2007.4381486`.

[3] Thawatchai Chomsiri and Chotipat Pornavalai. "Firewall Rules Analysis". In: Jan. 2006.

[4] Antonio Coscia, Vincenzo Dentamaro, Stefano Galantucci, Antonio Maci, and Giuseppe Pirlo. "An innovative two-stage algorithm to optimize Firewall rule ordering". In: *Computers & Security* 134 (Aug. 2023), p. 103423. DOI: `10.1016/j.cose.2023.103423`.

[5] Errin Fulp. "Optimization of Network Firewall Policies using Directed Acyclical Graphs". In: (Jan. 2005).

[6] K. Golnabi, R.K. Min, L. Khan, and E. Al-Shaer. "Analysis of Firewall Policy Rules Using Data Mining Techniques". In: *2006 IEEE/IFIP Network Operations and Management Symposium NOMS 2006.* 2006, pp. 305–315. DOI: `10.1109/NOMS.2006.1687561`.

[7] Mohamed Gouda and Alex Liu. "Structured firewall design". In: *Computer Networks* 51 (Mar. 2007), pp. 1106–1120. DOI: `10.1016/j.comnet.2006.06.015`.

[8] Graphiz. *Graphiz.* 2024. URL: `https://graphviz.org/`. (accessed: 2024.01.17).

[9] Digital Guardian. *Digital Guardian - What is Data Loss Prevention (DLP)?* 2023. URL: `https://www.digitalguardian.com/blog/what-data-loss-prevention-dlp-definition-data-loss-prevention`. (accessed: 2023.10.03).

[10] Dhwani Hakani. "A Survey on Firewall for cloud security with Anomaly detection in Firewall Policy". In: *2023 International Conference on Artificial Intelligence and Smart Communication (AISC).* 2023, pp. 825–830. DOI: `10.1109/AISC56616.2023.10085419`.

[11] Hazem Hamed and Ehab Al-Shaer. "On autonomic optimization of firewall policy organization". In: *J. High Speed Networks* 15 (Jan. 2006), pp. 209–227.

[12] Hongxin Hu, Gail-Joon Ahn, and Ketan Kulkarni. "Detecting and Resolving Firewall Policy Anomalies". In: *IEEE Transactions on Dependable and Secure Computing* 9 (May 2012), pp. 318–331. DOI: `10.1109/TDSC.2012.20`.

[13] Tihomir Katic and Predrag Pale. “Optimization of Firewall Rules”. In: July 2007, pp. 685–690. ISBN: 953-7138-10-0. DOI: 10.1109/ITI.2007.4283854.

[14] Bilal Khan, Maqsood Mahmud, Khurram Khan, and Khaled Alghathbar. “Security Analysis of Firewall Rule Sets in Computer Networks”. In: (July 2010). DOI: 10.1109/SECURWARE.2010.16.

[15] Ahmed Khoumsi, Mohammed Erradi, and Wadie Krombi. “A Formal Basis for the Design and Analysis of Firewall Security Policies”. In: *Journal of King Saud University - Computer and Information Sciences* 30 (Nov. 2016). DOI: 10.1016/j.jksuci.2016.11.008.

[16] Lepide. *The Role of Data Classification in Data Loss Prevention (DLP).* 2023. URL: https://www.lepide.com/blog/the-role-of-data-classification-in-data-loss-prevention-dlp/. (accessed: 2023.10.03).

[17] Simon Liu and Rick Kuhn. “Data Loss Prevention”. In: *IT Professional* 12.2 (2010), pp. 10–13. DOI: 10.1109/MITP.2010.52.

[18] Ehab Al-Shaer and Hazem Hamed. “Discovery of policy anomalies in distributed firewalls”. In: Apr. 2004, 2605–2616 vol.4. ISBN: 0-7803-8355-9. DOI: 10.1109/INFCOM.2004.1354680.

[19] ASHISH TAPDIYA. “FIREWALL POLICY OPTIMIZATION AND MANAGEMENT”. In: (Jan. 2008).

[20] Ashish Tapdiya and Errin Fulp. “Towards Optimal Firewall Rule Ordering Utilizing Directed Acyclical Graphs”. In: Sept. 2009, pp. 1–6. DOI: 10.1109/ICCCN.2009.5235232.

[21] Artem Voronkov, Leonardo A. Martucci, and Stefan Lindskog. “Measuring the Usability of Firewall Rule Sets”. In: *IEEE Access* 8 (2020), pp. 27106–27121. DOI: 10.1109/ACCESS.2020.2971093.

[22] M. Yoon, S. Chen, and Z. Zhang. “Reducing the Size of Rule Set in a Firewall”. In: *2007 IEEE International Conference on Communications.* 2007, pp. 1274–1279. DOI: 10.1109/ICC.2007.215.

Appendix A

Appendix

Listing A.1: Rule 2 implementation

```
<and>
  <!-- confidential -->
  <equal>
    <evtSrcFilePolicyTag/>
    <string value="confidential"/>
  </equal>
  <!-- companies website -->
  <in like='both'>
    <evtDomain/>
    <list>
      <string value="company.com"/>
      <string value="company-drive.ch"/>
      <string value="company-webmail.ch"/>
    </list>
  </in>
  <!-- fileUpload -->
  <equal>
    <evtOperationType/>
    <constOpNetTransferUpload/>
  </equal>
</and>
```