UNIVERSITÉ DE STRASBOURG

ÉCOLE DOCTORALE MATHÉMATIQUES, SCIENCES DE L'INFORMATION ET DE L'INGÉNIEUR

Laboratoire Icube – UMR7357

THÈSE présentée par :

Loïc Miller

soutenue le 22 avril 2022

pour obtenir le grade de : **Docteur de l'université de Strasbourg**

Discipline/ Spécialité : **Informatique**

Securing Workflows: On the Use of Microservices and Metagraphs to Prevent Data Exposures

Abstracts

Résumé

Cette thèse traite du sujet des fuites et des violations de données dans les workflows. Les workflows sont utilisés partout et par tout le monde, les workflows multi-organisations en particulier étant nécessaires chaque fois qu'il y a une collaboration entre deux ou plusieurs organisations. Les fuites et les violations de données sont généralisées, résultent régulièrement des vulnérabilités les plus critiques et se produisent de plus en plus. Le passage au cloud et aux approches modulaires a augmenté les surfaces d'attaque et apporté de nouveaux risques de sécurité qui n'existaient pas auparavant.

Compte tenu de cette situation, voyant de grandes entreprises stockant leurs données dans le cloud sans chiffrement pour établir leurs workflows multi-organisations, il existe un fort besoin d'une solution capable de déployer ces workflows en toute sécurité et d'éviter les fuites de données. Cette thèse porte donc sur la prévention des fuites et violations de données, notamment dans les workflows. Notre objectif est de permettre l'utilisation de workflows multi-organisations sécurisés et d'atténuer le risque de fuites de données.

Dans un premier axe, nous construisons une infrastructure utilisant les microservices qui permet l'exécution de workflows dans le cloud et empêche les fuites de données. Une des limitations de notre infrastructure est le fait que les politiques de sécurité doivent être correctes pour bénéficier de leur sécurité. Dans un deuxième axe, nous répondons à cette limitation en expliquant comment vérifier que la spécification d'une politique de sécurité corresponde à sa traduction en implémentation. Dans ce but, nous utilisons une construction qui généralise le graphe simple, les métagraphes. Dans le troisième axe, nous abordons le domaine adjacent de l'analysis de politiques, pour garantir que les spécifications de celles-ci sont correctes. En particulier, nous abordons le problème de la recherche de redondances dans la spécification d'une politique.

Mots clés: fuites de données, workflows, microservices, autorisation, contrôle d'accès, vérification de politiques, métagraphs, yawl, rego, hypergraphes, hypernetworks, complexité, NP-complétude

Abstract

This thesis deals with the topic of data leaks and breaches in workflows. Workflows are used everywhere and by everyone, multi-party workflows in particular being needed whenever there is a collaboration between two or more organizations. Data leaks and breaches are widespread, regular outcomes of the most critical vulnerabilities, and happening continuously more. The shift to the cloud and modular approaches has increased attack surfaces and brought new security risks that did not exist before.

Considering this situation, with major companies storing their data unencrypted in the cloud to enable their multi-party workflows, there is a strong need for a solution that can enable those workflows securely and prevent exposures. This thesis therefore focuses on the prevention of data exposures, in workflows in particular. Our goal is to enable secure multi-party workflows and mitigate the risk of data exposures.

In a first axis, we build an infrastructure using microservices that allows the execution of workflows in the cloud and prevents data leaks. One of the limitations of our infrastructure is the fact that security policies must be free from errors to benefit from their security. In a second axis, we deal with this limitation by explaining how to verify that the specification of a policy corresponds to its translation into a policy implementation. For this purpose, we use a construct that generalizes the simple graph, metagraphs. In the third axis, we address the adjacent area of policy analysis, to ensure that policy specifications are correct. In particular, we tackle the problem of finding redundancies in the specification of a policy.

Keywords: data leak, workflows, microservices, authorization, access control, policy verification, metagraphs, yawl, rego, hypergraphs, hypernetworks, complexity, NP-completeness

Colophon
This document was typeset with the help of KOMA-Script and LaTeX using the kaobook class.

The source code of this book is available at:
`https://github.com/fmarotta/kaobook`

Acknowledgments

A journey of more than three years is drawing to a close. Over those years, I been blessed with many encounters, and in this preface, I want to take the time to thank all the people who supported me and tagged along. This preface will never do justice to the immense gratitude I feel towards each and everyone of you, but I will nonetheless try my best.

First of all, I want to thank the members of the jury who accepted to evaluate my thesis. Many thanks to Etienne Rivière, Géraldine Texier, Gregory Blanc and Sébastien Tixeuil. Thank you to Matthew Roughan in particular. It really is a pleasure to exchange with you, and I hope we can continue to do so in the future. Thanks again to the members of the jury for reviewing this thesis, I hope it is an enjoyable read.

Next, I want to thank my supervisors. Thank you to Pascal Mérindol. You were the one that introduced me to the world of research, and I can't help but smile when I think back on the D-CART internship and our path combinatorics. Thank you for being there all those years, and for being as passionate as you are when discussing complexity and algo-related problems. I don't think I will ever get tired of our discussions, whether they are academic or not. I wish you the very best for your Habilitation à Diriger des Recherches, I'm sure you will succeed in your endeavors whatever they may be.

Thank you to Cristel Pelsser. From our BGP blackjacks to workflows, you were always supportive and willing. Thank you for being a great supervisor, and for all the effort you put in and continue to put in. You were always available when I needed you, and for that, I feel very grateful. Thanks as well for giving me the opportunity to present my works abroad, whether in Luxembourg, or in Italy for summer schools, or even for participating in the organization of IMC. Finally, I can't thank you for helping me find my path forward in research. Thank you for everything you taught me, it really means the world to me.

Thank you to Antoine Gallais. From our initial discussions to the present, your support has been invaluable. I regret that we could not meet more often with you being remote, but want to express my gratitude for all the advice you gave me over the years.

I also want to take some time to the other permanent members of the Network Research Team. Thank you to Quentin Bramas, Stéphane Cateloin, Pierre David, Anissa Lamani, Julien Montavont, Thomas Noel, Guillaume Schreiner and Fabrice Theoleyre, for the moments we shared and discussions we had. I extend my gratitude to Quentin Bramas in particular, for always putting up with my complexity problems, showing interest and always taking the time to explain things. Thank you to Fabrice Theoleyre for being a model when it comes to research. Your rigorousness and the extensiveness of your knowledge have always amazed me.

Thanks as well to the many PhD students of the lab. Thank you to Jean-Philippe for always bringing a good mood to the lab. Even if you support AC Milan, I still look forward to our chess games, as well as your next SDPP presentation of course. Thank you to Jean-Romain, for all our discussions and our collaborations. Thank you for always being there to help, and always being chill, even in the most dire circumstances (RIP mini-internet). Thank you to Thomas A. for your sense of humor and authenticity. Thank you to Thomas H. for our discussions and our bouldering sessions. Talking about research and other topics with you is always a pleasure. Thank you to Renato, for being the bedrock of the PhD student room and always helping those in need. Thank you to Farzad. I wish you all the best of luck for what the future holds.

Thanks as well to the former PhD students of the lab. Thank you to Amine for sometimes letting me win at chess, and being enthusiastic about new projects. I'm sure you will get that FIDE title, and I wish you the best for your space endeavors. Thank you to Julian, as well as Rodrigo, Odnan and Sebastian. You guys always brought positive vibes along with you.

Thank you to Max. From our initial exchanges to our weekly meetings, it is a joy to interact with you. Your insights have been invaluable, and I'm sure you will prevail in whatever you put your mind to. Thank you to Reynaldo. I can't thank you enough for coming to talk to me after that presentation at FOSAD. I really appreciate that you took the time to help me with those reductions. You transmitted your passion for this field to me, and for that, you have my deepest gratitude. I look forward to seeing you, Adrien, Tullio, and the guys again. Working with you both is a real pleasure, and I look forward very much to our continued collaboration. Though I would say most of our work is still unpublished, I have high hopes it will be in the future.

I now step away from the academic world for a few moments to thank all the people outside of it who supported me.

Thank you to Brandon. From the time we were doing our masters, we never were apart for too long. I look forward to hearing your adventures at a certain company. I wish you all the best for the future; I'm sure you'll find what you're looking for.

Many thanks as well to the usual gang. We've all known each other for more than 10 years. Thank you to Antoine, my New Zealander and associate against creatures made of clay. Thank you to Jordan, especially when he arm-wrestles. Thank you to Adrian. I want you to know it's okay to support the wrong reds; you'll come around eventually. See you soon at Papa Lisa for your favorite dish. Thank you to Quentin, who is always ready for another Holiday Special. Thank you to Rémy, whether we're in Jarny, Colmar or somewhere else. With you, I could start over at the yellows as many times as it takes. Thank you to Guillaume, and his (sketchy?) PT advice. Thank you to Léa, especially when you sing along Linkin Park. Thank you Luc, simply for being you. We've known each other since high school and I wouldn't change the world for it.

Thank you to Ugo. It's always a pleasure when you come visit, particularly when we get to play (buy?) a certain €30 game. Thank you to Val and Doris. We don't see each other that often, but you're in my heart nonetheless.

I also want to extend my thanks to family members.

Thank you to my uncle, Philippe, for always sharing his enthusiasm and knowledge about the blue and whites.

Thank you to Emma, my sister, even when she's invaded by her landlord.

Thank you to Sylvie, my mother, for always being supportive no matter what. You're perhaps the kindest person I know.

Finally, thank you to Daniel, my father. You showed and continue to show me what it means to be someone with integrity and values, and I couldn't be more thankful for that.

Contents

List of Figures

List of Tables

Résumé

1 Introduction

Dans le monde industrialisé d'aujourd'hui, la plupart des organisations ont besoin de l'exécution de *workflows* pour atteindre leur buts. Des sociétés de trading à haute fréquence jusqu'aux organisations à but non lucratif, ces groupes suivent des ensembles de tâches, qu'un workflow soit explicitement défini ou non. Ces tâches représentent des actions telles que l'expédition de produits (Figure 1), la mise à jour de bilans ou toute autre action imaginable.

La plupart des workflows de ces organisations contiennent de nombreuses parties en mouvement. De plus, ces workflows impliquent le plus souvent d'autres organisations, ce qui entraîne la création de workflows multipartis. Les workflows multipartis surviennent dans plusieurs scénarios, qu'il s'agisse de plusieurs entreprises collaborant vers un objectif commun, d'une entreprise engageant d'autres entreprises pour effectuer des tâches spécifiques, ou même d'entreprises virtuelles [1] . Une entreprise virtuelle [2, 3] représente un cluster d'acteurs économiques, qui collaborent, partagent des compétences et des ressources afin de répondre rapidement aux changements du marché pour exploiter des opportunités commerciales [4].

[1]: Polyantchikov et al. (2017), 'Virtual enterprise formation in the context of a sustainable partner network'

Les workflows multipartis représentent l'un des plus grands défis pour les entreprises actuelles. Une organisation dans un tel workflow peut avoir besoin de communiquer avec des acteurs à la fois internes et externes et peut ne pas avoir un accès direct à toutes les ressources et données nécessaires à l'exécution de ses tâches. Cela pose des complications évidentes du côté de la communication et de la gestion, et surtout du côté de la sécurité.

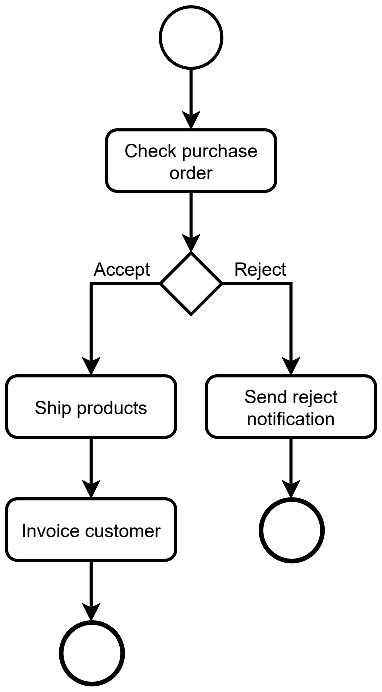


Figure 1: Workflow simple représentant la commande d'un client.

Workflows et pratiques de sécurité

La construction d'un workflow multipartis est très différente de la construction d'un workflow simple restant au sein d'une organisation. Ces premiers permettent à plusieurs organisations de participer à l'exécution des tâches dans le workflow.

Dans les workflows simples, les tâches et les processus sont construits autour de l'organisation qui le définit. Malheureusement, cela fait qu'au moment de créer un workflow multipartis, il en résulte souvent que ces tâches et processus sont toujours utilisés, avec par dessus un moyen supplémentaire de transférer des données entre les partis. Certaines entreprises utilisent le courrier électronique à cette fin, d'autres partagent des données via le cloud ou sur des marchés prévus à cet effet [5] .

[5]: European Commission et al. (2018), *Study on data sharing between companies in Europe*

Dans l'industrie cinématographique en particulier, les entreprises réalisant un film emploieront des sous-traitants pour effectuer des opérations de post-production sur les séquences du film. Ceux-ci incluent, sans s'y limiter, les effets spéciaux, la maîtrise du son et l'ajout de la technologie High Dynamic Range (HDR). Pour partager les séquences de film avec ces sous-traitants, l'entreprise possédant ces séquences stocke les données non chiffrées dans le cloud dans la grande majorité des cas [6] . Il s'agit évidemment d'une mauvaise pratique de sécurité, et pourrait expliquer en partie pourquoi les compromissions de données continuent de se produire.

[6]: Lifar et al. (2017), *Data Leak Prevention*

Compromissions de données

Les compromissions de données font référence à des incidents où des données sensibles censées être protégées sont consultées par une entité non autorisée. Les exemples de compromissions de données vont de simples erreurs révélant les données à quiconque les cherche [7] , à des attaquants exploitant des failles dans la sécurité d'un système pour accéder aux données [8] .

[7]: Krebs (2019), *First American Financial Corp. Leaked Hundreds of Millions of Title Insurance Records*

[8]: Stempel et al. (2017), *Yahoo says all three billion accounts hacked in 2013 data theft*

Les compromissions de données peuvent être scindées en deux catégories : les fuites de données et les violations de données. Même si dans ces deux cas les données sont consultées par des entités non autorisées, la manière dont ces données sont compromises est différente.

Les violations de données font référence à l'accès non autorisé aux données en exploitant des failles de sécurité du système compromis. Ces évènements peuvent se produire sur des données au repos [8] ou en transport [9].

Les fuites de données font référence à la compromission de données appartenant à une organisation, en raison de la manière dont ces données sont traitées par cette organisation. Elles peuvent se produire par erreur [7] ou par l'activité malveillante d'un(e) initié(e) [10].

Les compromissions de données sont critiques

La compromission de données est l'un des problèmes les plus critiques aujourd'hui. L'Open Web Application Security Project (OWASP), une fondation dédiée à l'amélioration de la sécurité des logiciels [11], dresse une liste des principales vulnérabilités affectant les applications Web aujourd'hui. Le *OWASP Top 10* [11] est basé sur un consensus d'experts en sécurité du monde entier, et est mis à jour tous les deux ou trois ans en fonction des évolutions de la sécurité des applications.

Arrivant en tête de cette liste comme la vulnérabilité la plus critique, le *contrôle d'accès imparfait* est passé de la cinquième place du Top 10 OWASP 2017 à la première dans l'itération la plus récente de la liste [12].

Les vulnérabilités courantes du contrôle d'accès incluent :

- La violation des principes de *least privilege* et de *deny by default*.
- Le contournement des vérifications de contrôle d'accès.
- L'élévation de privilèges, la manipulation ou l'exploitation d'une mauvaise configuration pour obtenir un accès qui aurait autrement été non autorisé.

Nous proposons des moyens d'améliorer le contrôle d'accès dans le Chapitre 4 et le Chapitre 5. Plus particulièrement, nous montrons comment vérifier que l'implémentation d'une politique correspond à sa spécification dans le Chapitre 4, et essayons de trouver et d'éliminer les redondances dans les spécifications de politiques dans le Chapitre 5.

Le deuxième de la liste OWASP, les *défaillances cryptographiques*, sont également très pertinentes pour nous. Cette catégorie traite des défaillances cryptographiques, qui conduisent souvent à la compromission de données sensibles. Les vulnérabilités courantes des défaillances cryptographiques incluent la transmission de données sans chiffrement, l'utilisation de protocoles cryptographiques obsolètes ou faibles, ou bien encore la réutilisation de clés.

Nous proposons des moyens d'atténuer ce type de vulnérabilités dans le Chapitre 3.

Les compromissions de données se produisent de plus en plus

Nous pouvons malheureusement observer que les fuites et les violations de données se produisent de plus en plus. Le nombre de compromissions et la quantité de données compromises ont tous deux suivi une tendance à la hausse [13–15] (Figure 2 et Figure 3). La quantité de données compromises[1] était de plus de 40 milliards en 2021 [13], contre 37,2 milliards pour 3 932 incidents en 2020 [14] et 15,4 milliards pour 7 553 incidents en 2019 [15]. C'est bien plus qu'il n'y en avait une décennie plus tôt, où le nombre de compromissions était de 2 644 et la quantité de données compromises était de 267 millions [16]. *Risk Based Security* précise dans son rapport de 2020 [14] que le piratage a été responsable de la plupart des incidents, mais que la compromission accidentelle de données sur Internet (par exemple, des bases de données, des sauvegardes, des services mal configurés) a mis le plus de données en danger.

Le nombre de violations de données en particulier est important et affecte un grand nombre d'organisations telles que les agences gouvernementales, les commerces, les agences de crédit, les studios de cinéma et bien d'autres [6].

La prévention des deux formes de compromission de données est un problème difficile à résoudre, car comme nous l'avons vu, une compromission peut se produire dans de multiples circonstances, causée par un piratage, des bases de données mal configurées, des initiés malveillants, etc. Notez que ces compromissions sont perçues comme d'énormes pertes d'argent pour des entreprises comme celles de l'industrie cinématographique [17] , et comme une perte de confidentialité pour les applications traitant des données utilisateur [18].

[17]: Byers et al. (2003), 'Analysis of security vulnerabilities in the movie production and distribution process'

1: La quantité de données se mesure en nombre d'entrées dans une base de données

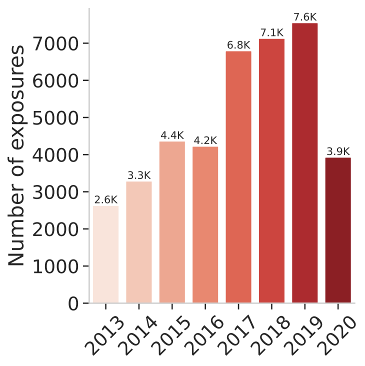


Figure 2: Nombre de compromissions signalées chaque année de 2013 à 2020 [14].

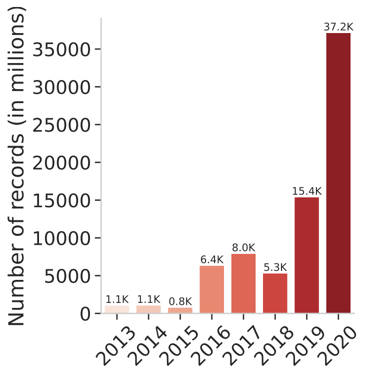


Figure 3: Quantité de données compromises chaque année (en millions) de 2013 à 2020 [14].

2 Contributions de cette thèse

Dans la section précédente, nous avons dressé un portrait succint de la situation actuelle. Les workflows sont utilisés partout et par tout le monde, les workflows multipartis étant nécessaires chaque fois qu'il y a une collaboration entre deux organisations ou plus.

Les fuites et les violations de données sont généralisées, résultent des vulnérabilités les plus critiques et se produisent de plus en plus. De plus, le passage au cloud et aux approches modulaires a augmenté les surfaces d'attaque et apporté de nouveaux risques de sécurité qui n'existaient pas auparavant.

Compte tenu de cette situation, avec de grandes entreprises stockant leurs données non chiffrées dans le cloud pour exécuter leurs workflows multipartis, il y un besoin fort pour solution capable d'exécuter ces workflows en toute sécurité et d'éviter les compromissions de données.

Cette thèse porte donc sur la prévention des compromissions de données, notamment dans les workflows. Notre objectif est de permettre l'exécution de workflows multipartis sécurisés et d'atténuer le risque de compromissions de données.

Les vulnérabilités les plus critiques à l'origine des compromissions de données sont le *contrôle d'accès imparfait* et les *défaillances cryptographiques*. Conformément à ce fait, nous avons structuré cette thèse autour de trois axes qui tentent d'atténuer ces vulnérabilités.

Dans le Chapitre 3, nous proposons une infrastructure utilisant des microservices pour permettre des workflows multipartis sécurisés et gérer les compromissions de données. Nous réalisons une preuve de concept de cette infrastructure, et la déployons sur Google Cloud Platform. Ce chapitre se concentre davantage sur les défaillances cryptographiques, car nous donnons des moyens de sécuriser les données au repos ainsi qu'en transport.

- ► *RQ1: Comment pouvons-nous construire un workflow multipartis sans fuite ?*

Le Chapitre 4 et le Chapitre 5 s'occupent du contrôle d'accès imparfait. Étant donné que les systèmes à sécuriser par contrôle d'accès peuvent être très complexes, les administrateurs s'appuient souvent sur une gestion du contrôle d'accès basée sur des politiques. Les politiques définissent le comportement souhaité d'un système d'un point de vue de haut niveau. Par conséquent, cette forme de gestion permet de séparer le problème de spécification, c'est-à-dire la définition du comportement souhaité du système, du problème de l'implémentation, c'est-à-dire l'application du comportement souhaité du système. Cela rend la gestion du contrôle d'accès plus facile et plus flexible. Pour donner un exemple, une spécification de politique peut consister en une matrice de contrôle d'accès, un graphe de privilèges ou un document YAWL, tandis qu'une implémentation de politique peut être une liste de contrôle d'accès, un document XACML ou encore un document Rego.

Dans le Chapitre 4, nous nous sommes intéressés à la vérification de politiques de sécurité déployées. Nous étudions la gestion du contrôle d'accès basé sur des politiques, et plus spécifiquement les moyens de vérifier qu'une spécification de politique correspond à son implémentation. Nous développons un outil permettant la vérification des politiques à l'aide de métagraphes, une structure théorique généralisée des graphes. Les implémentations ne sont généralement pas exemptes d'erreurs, et un mécanisme de vérification peut atténuer le risque d'échec du contrôle d'accès.

- *RQ2: Comment vérifier qu'une spécification de politique correspond à son implémentation déployée ?*

Dans le Chapitre 5, nous traitons de l'analyse des politiques. En particulier, nous analysons les spécifications de politiques pour détecter et éliminer les redondances potentielles. Nous proposons un moyen de détecter les redondances, et développons un outil pour le faire. En supprimant l'encombrement des spécifications de politique, nous espérons donner aux administrateurs de politique une vue plus claire de celle-ci. De cette façon, nous pouvons les aider à faire moins d'erreurs lorsqu'ils traitent une spécification, et donc à terme réduire le risque d'erreur au niveau du contrôle d'accès.

- *RQ3: Comment vérifier qu'une spécification ne contient aucune redondance ?*

Nous résumons nos contributions dans la Table 1.

Table 1: Résumé de nos contributions.

Chapitre	Sujet	Publication
Chapitre 3	Infrastructure sécurisée	[19, 20]
Chapitre 4	Vérification de politiques	[20, 21]
Chapitre 5	Elimination de redondances	[22]

Nous avons également développé plusieurs outils propres à chaque contribution. Nous résumons les outils que nous avons développés dans la Table 2.

Table 2: Résumé de nos outils.

Chapitre	Outil	Dépôt
Chapitre 3	Preuve de Concept	`https://github.com/loicmiller/secure-workflow`
Chapitre 4	Vérification de politiques	`https://github.com/loicmiller/policy-verification`
Chapitre 4	MGToolkit pour Python 3	`https://github.com/loicmiller/mgtoolkit`
Chapitre 5	Elimination de redondances	`https://github.com/loicmiller/policy-analysis`
Chapitre 5	Formulation SAT	`https://github.com/loicmiller/fhep-sat-formulation`

3 Une infrastructure sécurisée pour empêcher les compromissions de données

Pour atténuer le risque de compromission des données, nous proposons dans le Chapitre 3 une infrastructure utilisant l'architecture microservices. Plus particulièrement, nous fournissons des moyens de sécuriser les données au repos et en transport, ainsi qu'un moyen pour le gestionnaire du workflow d'appliquer une politique de sécurité.

Énoncé du problème

Dans le cadre d'un workflow, le système que l'on souhaite réaliser doit permettre l'échange sécurisé des données et sa sécurité au repos tout en évitant les fuites.

Nous définissons un workflow comme une séquence de tâches à effectuer par un ensemble d'acteurs indépendants. Le propriétaire des données (c'est-à-dire l'instigateur du workflow) interagit avec les sous-traitants pour réaliser une telle séquence. Le workflow est défini par le propriétaire, qui définit comment et par qui les données qu'il possède doivent être traitées. Il précise également les différentes étapes nécessaires pour atteindre son objectif. Ces tâches sont réalisées par des sous-traitants, qui exécutent la tâche qui leur a été confiée, sur les données qui leur ont été fournies. Le propriétaire et le sous-traitant ont tous deux des agents qui traitent les données, où les agents peuvent représenter soit un employé soit un service automatique.

Prenons le cas d'un workflow, où un propriétaire en phase de post-production d'un film souhaite employer d'autres sociétés pour éditer les composants vidéo et audio du film [17]. Plus précisément, imaginons que, par exemple, le propriétaire souhaite ajouter des effets spéciaux, régler les couleurs, configurer la technologie *High Dynamic Range* (HDR) et faire un mastering de l'audio. En particulier, il souhaite d'abord l'application des effets spéciaux, puis le réglage des couleurs et, enfin, le HDR en parallèle avec le mastering du son.

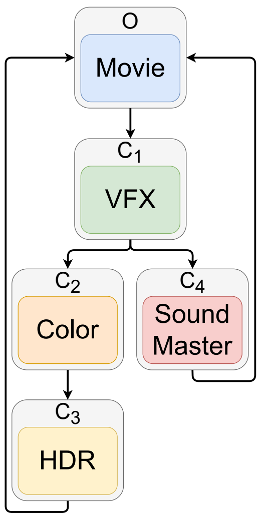


Figure 4: Exemple de workflow de film. Les flèches modélisent le workflow spécifié et représentent ainsi le flux de communication de notre exemple. Le propriétaire (O) envoie ses données au premier sous-traitant (C_1), pour le traitement des effets spéciaux. C_1 envoie ensuite les données modifiées tout au long du workflow, pour le traitement des couleurs (C_2), HDR (C_3) et du son (C_4). Les données résultantes sont ensuite renvoyées au propriétaire.

L'intention du propriétaire peut être modélisée sous la forme d'un workflow, c'est-à-dire d'un graphe orienté acyclique tel que représenté sur la Figure 4. Le propriétaire (O) envoie d'abord ses données à la société responsable des effets spéciaux (C_1). C_1 applique des effets spéciaux aux séquences de films que le propriétaire lui a envoyées, puis envoie le résultat à la société responsable de la coloration (C_2) ainsi qu'à une autre pour le mastering du son (C_4). C_2 envoie ensuite son résultat à l'agent en charge du High Dynamic Range (HDR) (C_3). Enfin, C_3 et C_4 renvoient leur sortie au propriétaire, qui rassemble les données pour obtenir le produit final..

Le but de notre solution est de permettre à ce workflow de se dérouler, le film transitant du propriétaire (O) en passant par les sous-traitants (C_1, C_2, C_3, C_4) jusqu'au propriétaire, tout en garantissant la sécurité des données au repos et en transport. Avec notre proposition, nous empêchons les communications indésirables au niveau de la couche réseau et applicative.

Description générale de l'infrastructure

Comme nous avons besoin d'un moyen d'empêcher les fuites de données, nous devons contrôler les communications auxquelles un agent peut s'engager. Pour y parvenir, nous devons contrôler les environnements que les agents utiliseront, pour nous assurer que toutes les actions d'un agent suivent une politique appliquée par le propriétaire. Nous avons choisi de le faire en utilisant l'architecture microservice, pour les avantages accordés par celle-ci. En outre, les microservices sont déjà couramment déployés et fournissent de nombreux services comme le passage à l'échelle et l'isolation.

Dans cette infrastructure, les agents de notre workflow sont mappés sur des conteneurs, qui sont ensuite utilisés conjointement avec un orchestrateur, un *service mesh* et des *policy engines* pour appliquer la politique du propriétaire.

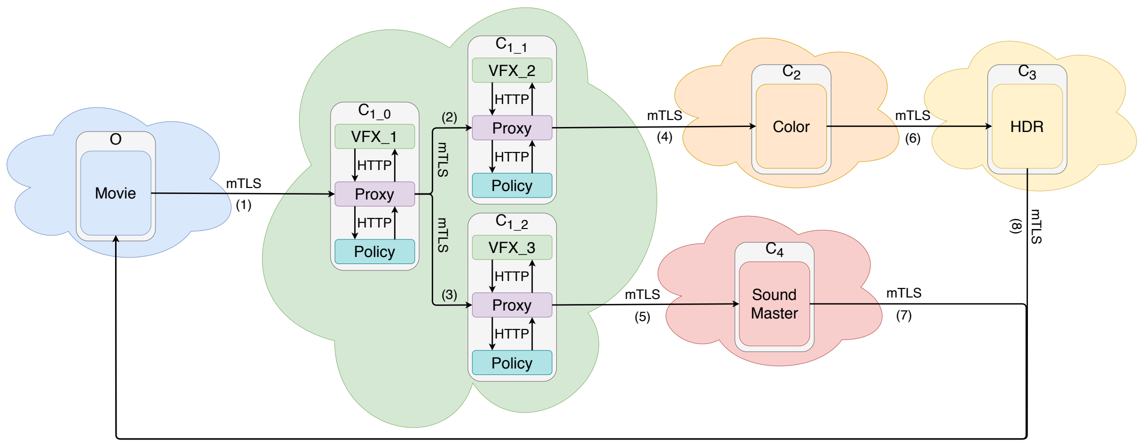


Figure 5: Infrastructure sécurisée. Chaque boîte représente un agent. C'est un pod avec les conteneurs appropriés. Le conteneur de la couleur de l'acteur représente le service. Les conteneurs violets représentent les proxys du service mesh et les conteneurs bleus représentent les sidecars de politiques. Les flèches précisent si les communications sont sécurisées (`mTLS`) ou non (HTTP).

La Figure 5 montre le workflow que nous avons défini dans la Figure Figure 4, chaque acteur ayant son propre espace de déploiement représenté par le nuage entourant les boîtes qui représentent les agents de ces acteurs (par exemple, la boîte C_{1_1} représente un agent de l'entrepreneur C_1). Les politiques d'accès d'un service sont poussées dans le sidecar de politiques associé au service.

La Figure 5 illustre également comment nous utilisons les éléments de l'architecture microservice. Chaque agent est un pod contenant le service (c'est-à-dire l'environnement que l'agent utilisera), un proxy et un sidecar de politiques. Le sidecar proxy interceptera tout le trafic provenant et allant vers son service. Le proxy va alors vérifier grâceau sidecar de politiques si la requête est autorisée ou non. Si la demande est autorisée, elle est transmise en conséquence, et la demande est rejetée dans le cas contraire.

Preuve de concept

Nous avons réalisé une preuve de concept, en implémentant l'infrastructure décrite. Nous reproduisons le workflow de la Figure 5, avec des services du workflow recevant et envoyant des données arbitraires pour représenter les données du propriétaire. Nous utilisons Docker [23] pour nos conteneurs, Kubernetes [24] pour notre couche d'orchestration, Istio [25] comme service mesh, en utilisant Envoy [26] pour les sidecars proxy et Open Policy Agent [27] (OPA) pour les sidecars de politiques. Nous utilisons également Kubernetes pour donner aux services des volumes chiffrés. Cette infrastructure a été déployée sur Google Cloud Platform (GCP), en utilisant un cluster pour chaque acteur du workflow, pour un total de cinq clusters.

Evaluation de performances

En mesurant le coût des mécanismes de contrôle d'accès, nous constatons que les pods avec Open Policy Agent ont une augmentation substantielle de leur temps de démarrage de près de deux secondes en moyenne

(32,72%). En mesurant le délai pour une requête en fonction de la taille de la politique, nous constatons que la taille de la politique représente 65% de la variance des communications intra-régionales alors qu'elle ne représente que 7% de la variance des communications interrégionales.

Ce travail a conduit à deux publications [19, 20] et au développement d'une Preuve de Concept accessible publiquement.

Notre infrastructure suppose que la spécification de la politique correspond à son implémentation. En réalité, cela peut être erroné et pourrait conduire à des attaques potentielles exploitant une politique défectueuse.

4 Vérification de politiques à l'aide de métagraphes

Pour abandonner cette supposition, nous nous sommes occupés dans le Chapitre 4 de la vérification des politiques. Dans ce chapitre, nous détaillons un moyen de vérifier que les politiques déployées dans notre infrastructure correspondent bien à leur spécification initiale.

Dans ce but, nous utilisons les métagraphes. Les métagraphes sont un outil de spécification efficace pour modéliser les processus et leurs workflows.

Nous pouvons pleinement exprimer YAWL, un langage de spécification de workflows, avec des métagraphes. Nous pouvons en d'autres termes exprimer comment chaque tâche, condition et opérateur peut être converti en une représentation métagraphique, soutenant la richesse du langage YAWL. Nous utilisons YAWL et Rego comme langages de référence respectivement pour la spécification et l'implémentation de la politique. Nous définissons une méthode permettant la vérification des politiques à l'aide de métagraphes.

En modélisant sous forme de métagraphes la spécification de la politique haut niveau, ainsi que l'implémentation de la politique traduite, non seulement nous pouvons rechercher des conflits et des redondances

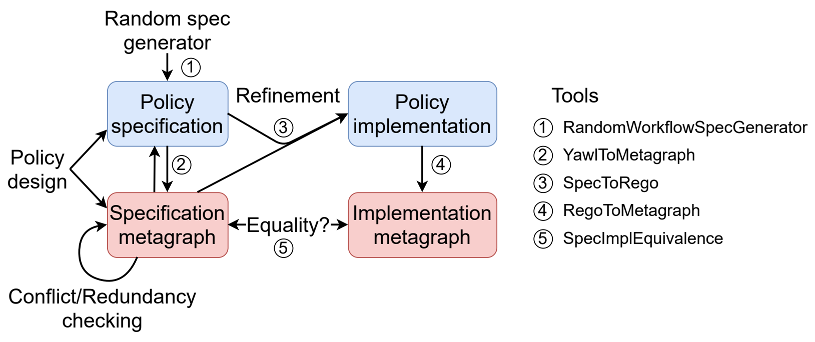


Figure 6: Vérification de politiques à l'aide de métagraphes. Lors de la conception de la politique, un administrateur de politique peut soit choisir de créer sa spécification de politique en représentant visuellement la politique sous la forme d'un métagraphe conditionnel, soit choisir de spécifier la politique dans un autre format. Si les métagraphes de spécification et d'implémentation sont égaux, nous concluons que l'implémentation de la politique correspond à la spécification de la politique. Ils ne correspondent pas si les métagraphes sont différents : nous en concluons que des erreurs de déploiement se produisent. De plus, et de par sa conception, le métagraphe de spécification peut être utilisé pour vérifier les redondances et les conflits dans la politique aux deux niveaux (spécification de haut niveau et déploiement).

aux deux niveaux [28], mais nous pouvons également comparer les deux afin de suivre le déploiement les erreurs. Si les métagraphes de spécification et d'implémentation sont égaux, ils correspondent alors (l'implémentation de la politique correspond à la spécification de la politique) et aucune erreur ne se produit, mais lorsqu'ils ne correspondent pas, les métagraphes ne sont pas équivalents – cela signifie que des erreurs se sont produites lors du raffinement et /ou déploiement. La Figure 6 résume notre construction globale, ainsi que les outils que nous avons développés pour réaliser la vérification.

Nous avons évalué notre méthode de vérification et constaté qu'elle s'exécutait dans un délai très raisonnable, même avec des politiques relativement importantes.

Ce travail a donné lieu à deux publications [20, 21], le développement d'une Version Python 3 de MGToolkit et le développement d'un framework de vérification des politiques. Ces outils sont accessibles publiquement.

Notre méthode de vérification comportait également certaines hypothèses. À savoir, nous supposons que la spécification de la politique est exempte d'erreurs et de redondances. C'est loin d'être toujours le cas. Notre méthode de vérification ne peut que conclure que l'implémentation d'une politique correspond à sa spécification, mais ne permet pas de détecter les erreurs et les redondances dans la spécification uniquement.

5 Analyse de politiques pour l'identification de redondances

C'est pourquoi dans le Chapitre 5, nous avons abordé le problème de la recherche de redondances dans une politique. Nous avons montré que la solution actuelle est inadéquate et prouvé que trouver des redondances dans un métagraphe est NP-difficile. Pour ce faire, nous avons utilisé des concepts venant de la littérature sur les hypergraphes. Sur la base de cette découverte, nous avons formulé notre problème comme un problème SAT pour avoir un moyen plus efficace de trouver des redondances, mais nous avons trouvé cette méthode inefficace.

Comme alternative, nous avons proposé un algorithme exhaustif utilisant le triangle de Pascal pour trouver les redondances, qui est plus efficace que la solution précédente. Pour confirmer ce résultat, nous avons mené une analyse de performance afin de comparer ces différentes méthodes de recherche de redondances. Comme prévu, notre méthode est plus rapide et plus évolutive que les autres. La Figure 5.11 décrit les temps d'exécution des algorithmes d'analyse de politique mesurés, en fonction du nombre d'arêtes dans la spécification du workflow. Globalement, nous constatons que l'algorithme utilisant le triangle de Pascal est bien le plus efficace des algorithmes testés, puisqu'il détecte les redondances, a une plus grande capacité en termes de taille de métagraphe, et de meilleurs temps d'exécution que les algorithmes d'énumération et SAT.

Une partie de ce travail, la preuve de complexité sur les F-hypergraphes acycliques, est actuellement en cours de soumission [22]. Nous avons

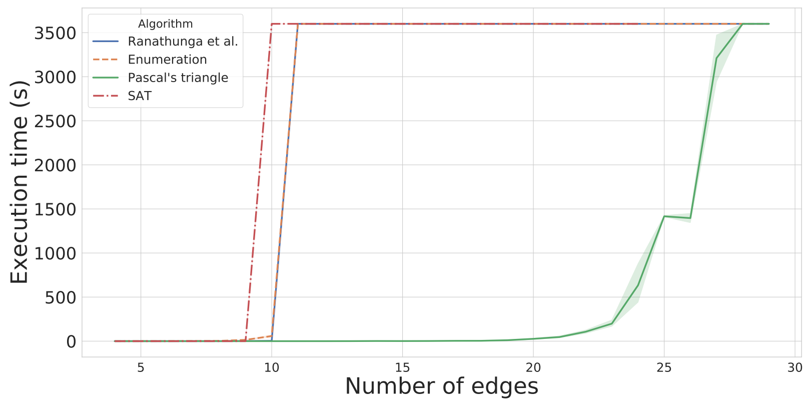


Figure 7: Temps d'exécution de différents algorithmes d'analyse de politiques. En particulier, nous comparons Ranathunga *et al.* [28], l'algorithme énumérant tous les métachemins, l'algorithme SAT et l'algorithme utilisant le triangle de Pascal.

également développé des outils accessibles au public, à savoir la formulation SAT et un framework d'analyse de politiques contenant les autres méthodes de détection de redondances présentées dans ce chapitre[2] . Le code pour générer des workflows, le code pour effectuer les mesures et autres fonctionnalités associées sont également disponibles.

2: L'algorithme de Ranathunga *et al.*, l'algorithme d'énumération et l'algorithme utilisant le triangle de Pascal.

6 Conclusion

Les principales conclusions de cette thèse sont les suivantes :

- **L'architecture microservice peut être utilisée pour construire un workflow multipartis sans fuites.**
  Nous montrons comment utiliser l'architecture microservice pour sécuriser les workflows. Nous définissons une infrastructure sécurisée et la déployons sur Google Cloud Platform dans une preuve de concept accessible publiquement. Avec cette infrastructure, nous espérons atténuer les compromissions de données en traitant les défaillances cryptographiques et en fournissant un moyen d'appliquer une politique de sécurité. Notre infrastructure ne protège cependant pas contre toutes les attaques, et a besoin de quelques hypothèses. Nous supposons en particulier que la spécification de la politique est correcte et correspond à son implémentation.

- **Les métagraphes sont une structure utile pour modéliser les politiques de workflow et peuvent être utilisés pour effectuer de la vérification de politiques.**
  A notre connaissance, c'est la première fois que quelqu'un montre comment vérifier des politiques à l'aide de métagraphes. Les métagraphes sont un moyen pertinent et efficace de modéliser, gérer et vérifier les politiques déployées, notamment les politiques de workflow. Nous avons montré que nous pouvons pleinement exprimer YAWL avec des métagraphes, profitant ainsi la richesse de ce langage en métagraphes. En vérifiant que la spécification correspond à l'implémentation, nous espérons réduire le nombre d'implémentations erronées et ainsi atténuer les expositions aux données dues à un contrôle d'accès imparfait.

- **Les métagraphes sont une structure utile pour analyser les politiques, en particulier pour identifier et supprimer les redondances.**

  La façon actuelle de détecter les redondances avec les métagraphes présente quelques problèmes. Nous avons montré que l'analyse d'une politique pour trouver des redondances est NP-difficile dans le cas général, et présenté un algorithme utilisant le triangle de Pascal. Nous soutenons qu'il s'agit d'une solution efficace à ce problème et confirmons que cette méthode est meilleure que d'autres moyens de trouver des redondances en procédant à une évaluation des performances. Nous espérons qu'en fournissant des moyens d'identifier les redondances, nous pourrons réduire davantage le risque de contrôle d'accès imparfait dans les politiques déployées.

Tous les outils, les données, le code permettant de générer des mesures, les résultats ainsi qu'un tutoriel, sont accessibles publiquement dans leur dépôt associé. Les liens vers ceux-ci peuvent être trouvés dans la Table **??**. Les données pour les mesures du Chapitre 4 prennent trop de place pour un dépôt git et peuvent être trouvées ici à la place.

Perspectives et futurs travaux

Pour le Chapitre 3, les travaux futurs incluent l'étude de l'impact des modifications du workflow sur la sécurité de notre infrastructure, ou la suppression de certaines hypothèses sur la confiance accordée à certaines entités. Une piste possible pour réduire ou supprimer ces hypothèses serait d'utiliser des *Trusted Execution Environments* (TEE), qui sont une zone sécurisée à l'intérieur d'un processeur.

Concernant le Chapitre 4, les travaux futurs possibles incluent l'extension des types de politiques que notre vérificateur peut gérer. Pour étendre le domaine des politiques que nous sommes en mesure de vérifier, une piste possible est d'étudier des solutions de correspondance de symboles afin de fournir une manière plus générale d'interpréter les identifiants dans ces politiques. Une autre piste importante est la construction d'un ensemble représentatif de données politiques accessibles publiquement.

En ce qui concerne le Chapitre 5, les travaux futurs possibles incluent la recherche d'une formulation SAT du problème différente, qui n'a pas besoin d'un nombre exponentiel de clauses. Une piste à essayer est l'application manuelle des lois de De Morgan, qui devrait réduire le temps de conversion des clauses en leur forme normale conjonctive. De plus, la complexité de notre solution actuelle utilisant le triangle de Pascal est malheureusement encore exponentielle. Pour atténuer cela et améliorer notre algorithme, nous pourrions utiliser une version de l'algorithme de Tarjan adaptée aux métagraphes afin de ne considérer que les arêtes sur un chemin de la source à la destination, puisque les autres arêtes ne feront pas partie d'un métachemin dominant. Dans un métagraphe, nous définissons une composante fortement connexe (SCC) comme un sous-ensemble de variables où chaque variable du sous-ensemble peut atteindre d'autres variables du sous-ensemble en utilisant un chemin simple. Pour trouver facilement des arêtes sur un chemin de la source à la destination, nous pourrions ajouter une arête virtuelle de la destination vers la source, puis exécuter l'algorithme de Tarjan pour trouver les SCC. La SCC contenant la source et la destination contiendra les arêtes à considérer lors de l'exécution de l'algorithme du triangle de Pascal. Étant donné que la SCC contiendra moins d'arêtes que le métagraphe complet, nous pouvons probablement gagner un peu de performance de cette façon. Alternativement, nous pourrions également effectuer un parcours en profondeur à partir de la source pour trouver les nœuds accessibles à partir de la source, et un parcours en profondeur à partir de la destination dans le métagraphe transposé pour trouver les nœuds accessibles à partir de la destination. Ensuite, l'intersection de ces deux ensembles donne les nœuds sur un chemin allant de la source à la destination.

D'autres travaux futurs possibles incluent l'utilisation de métagraphes/hypergraphes pour d'autres problèmes d'analyse des politiques. De nombreuses propriétés n'ont pas encore été explorées avec ces constructions, comme la *separation of duties* ou le *principle of least privilege*. Un autre de ces aspects est la détection de l'incomplétude de la politique, où l'on essaie d'identifier les requêtes qui n'ont pas de décision explicite.

Dans l'ensemble, nos objectifs à court terme incluent la recherche d'autres moyens de modéliser le problème d'analyse de politiques en tant que SAT/ILP. Nous avons seulement prouvé que ce problème était NP-difficile, mais il pourrait appartenir à une classe de complexité supérieure dans la hiérarchie, il pourrait donc y avoir une autre preuve à écrire dans ce but. Les objectifs à moyen terme incluent la recherche de l'impact de plusieurs patterns de workflow sur nos contributions. L'annulation de tâche en particulier semble être un modèle intéressant à étudier. Nous prévoyons également de continuer à explorer les résultats de complexité dans les métagraphes et les hypergraphes. Par exemple, nous pensons que le problème connexe de trouver des s-hyperréseaux peut être résolu plus rapidement pour certains types d'hypergraphes, et la complexité de cette procédure doit encore être déterminée pour d'autres types d'hypergraphes. Enfin, les objectifs à plus long terme incluent la constitution d'un ensemble représentatif de politiques accessibles publiquement. Nous ne sommes pas les seuls à penser qu'il s'agit d'une partie manquante cruciale du corpus de recherche actuel. Nous prévoyons également d'enquêter davantage sur les problèmes liés aux politiques et d'essayer de les résoudre à l'aide de métagraphes. Cela inclut des propriétés de politiques telles que la vérification de la *separation of duties* ou la détection d'incomplétude dans les politiques. Un aspect qui mériterait également une enquête plus approfondie est l'influence des patterns de workflows, comme l'annulation, sur nos différentes contributions. À l'avenir, nous espérons que le nombre croissant de travaux sur les politiques, les workflows et les métagraphes sera une incitation suffisante pour les administrateurs de politiques à les utiliser dans la pratique.

Introduction 1

1.1 Businesses and operations

In today's industrialized world, most organizations require the execution of workflows to attain their goal. From high-frequency trading firms all the way to nonprofit organizations, those groups follow sets of tasks whether a workflow is explicitly defined or not. Those tasks represent actions like shipping products (Figure 1.1), updating balance sheets, or any other action one can think of.

Most of those organizations' workflows contain many moving parts. Moreover, those workflows more often than not involve other organizations, resulting in multi-party workflows. Multi-party workflows arise in multiple scenarios, be it multiple businesses collaborating towards a common goal, one company hiring other businesses to perform specific tasks, or even virtual enterprises [1] . A virtual enterprise [2, 3] represents a cluster of economic actors, which collaborate, share skills and resources in order to respond rapidly to market changes to exploit business opportunities [4].

[1]: Polyantchikov et al. (2017), 'Virtual enterprise formation in the context of a sustainable partner network'

Multi-party workflows are one of the greatest challenges for current businesses. A single organization may need to communicate with actors both on the inside and outside, and may not have direct access to all the resources and data needed to perform their tasks. This poses obvious complications on the side of communication and management, and most of all on the side of security.

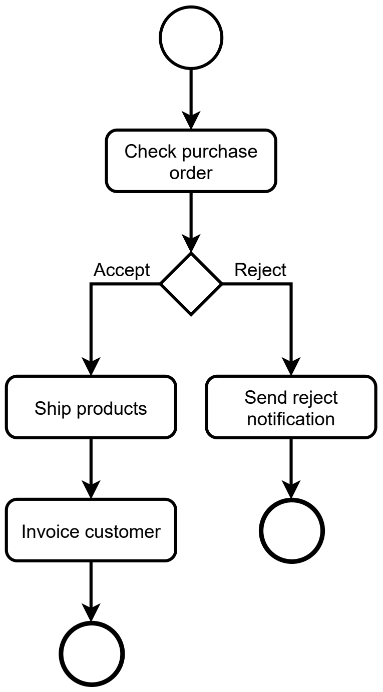


Figure 1.1: Simple workflow representing a customer order.

Workflows and security practices

Constructing a multi-party workflow is critically different than constructing a workflow staying within an organization. They allow multiple organizations to take part in the execution of tasks in the workflow.

In single-party workflows, tasks and processes are built around this single party. Regrettably, this makes it so when the time to construct a multi-party workflow comes, it often results in those single-party tasks and processes still being in use, with an additional way to transfer data between parties on top of it. Some companies use email to this end, some others share data using the cloud or marketplaces [5] .

[5]: European Commission et al. (2018), *Study on data sharing between companies in Europe*

In the movie industry most notably, businesses trying to make a movie will employ contractors to realize post-production operations on the movie footage. Those include but are not limited to special effects, sound mastering and adding High Dynamic Range (HDR) technology. To share the movie footage with those contractors, the business responsible for the footage will store the data unencrypted within the cloud, in the vast majority of cases [6] . This is of course a terrible security practice, and might explain in part why data exposures keep happening.

[6]: Lifar et al. (2017), *Data Leak Prevention*

1.2 Data exposures

Data exposures refer to incidents where sensitive data meant to be protected is accessed by an unauthorized party. Example of data exposures range from simple mistakes revealing the data to anyone that looks for it [7] to attackers exploiting flaws in the security of a system to access the data [8] .

[7]: Krebs (2019), *First American Financial Corp. Leaked Hundreds of Millions of Title Insurance Records*

[8]: Stempel et al. (2017), *Yahoo says all three billion accounts hacked in 2013 data theft*

Data exposures belong to one of two categories of events: data leaks and data breaches. Even though both data breaches and data leaks result in data being exposed to unauthorized entities, the way these data are exposed is different.

Data breaches

Data breaches refer to the unauthorized access of data by exploiting flaws in the security of the breached system.

Data breaches can happen with data at rest, where attackers exploit a flaw to gain access to the data. One of the most well-known examples of this is the 2013 Yahoo data theft [8]. Poor security practices at the company combined with a phishing attack allowed the perpetrators to gain access to the entirety of 3 billion Yahoo accounts. Namely, the attackers acquired names, email addresses, phone numbers, birth dates as well as hashed passwords and security questions and answers for all the accounts, which were protected with outdated easy-to-crack encryption. The breach was only reported by end of September 2016, and spawned multiple class action lawsuits.

Data breaches can also happen with data in transport, where attackers exploit a vulnerability to eavesdrop on traffic. For instance, consider the 2018 hijack of MikroTik routers [9] . Using vulnerability CVE-2018-14847, attacks gained access to all data being forwarded by more than 7,500 MikroTik routers, with no need for authentication. Note that the affected routers were part of core networks, and not simple home routers, so the amount of information that passes through them is much greater.

[9]: Seals (2018), *Thousands of MikroTik Routers Hijacked for Eavesdropping*

Data leaks

Data leaks refer to the exposure of data belonging to an organization, due to the way these data are processed by this organization.

Data leaks can happen by mistake. For example, the 2019 First American Financial Corporation leak comes to mind [7]. First American Financial Corporation is a financial services company providing title insurance and settlement services to the real estate and mortgage industries. In May 2019, a real estate developer in Washington reported FirstAm was leaking hundreds of millions of records through its website. Anyone who knew the URL for a valid document could access it by modifying the link, giving them access to mortgage deals going back to 2003. The leaked records included bank account numbers and statements, mortgage and tax records, as well as Social Security numbers, wire transaction receipts and drivers license images.

Data leaks can also be provoked by malicious behavior, where an insider leaks information to the public or an unauthorized third party. One such case happened in 2019 at Google, where an employee was reportedly fired for providing names and personal information of other employees to reporters [10] .

[10]: Lecher (2019), *Google reportedly fires staffer in media leak crackdown*

Data exposures are critical

Data exposures are one of the most critical issues today. The Open Web Application Security Project (OWASP), a foundation dedicated to improving the security of software [11], keeps a list of the top vulnerabilities affecting web applications today. The aptly named OWASP Top 10 [11] is based on a consensus from security experts around the world, and is updated every two or three years in accordance with changes in application security. The vulnerabilities are ranked based on their incidence rate, the potential impact they can have if exploited and their severity[1] .

1: The severity is estimated based on the incidence rate and impact of the vulnerability.

The most recent OWASP Top 10 was finalized in 2021 (Figure 1.2), and reflects the most critical vulnerabilities to date. Listed at the top as the most critical vulnerability, *Broken Access Control* has moved up from the 2017 OWASP Top 10 from the fifth place to the first [12].

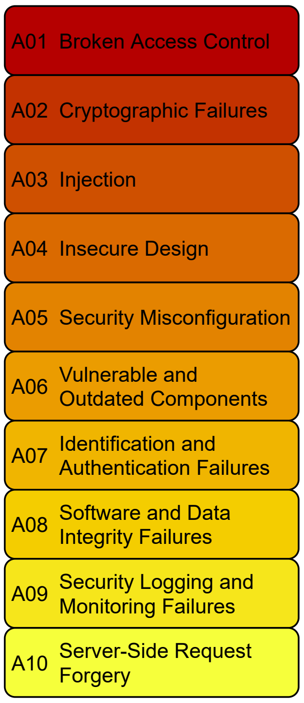

A01 Broken Access Control
A02 Cryptographic Failures
A03 Injection
A04 Insecure Design
A05 Security Misconfiguration
A06 Vulnerable and Outdated Components
A07 Identification and Authentication Failures
A08 Software and Data Integrity Failures
A09 Security Logging and Monitoring Failures
A10 Server-Side Request Forgery

Figure 1.2: OWASP Top 10 [11]. Vulnerabilities ranked based on frequency, severity and magnitude.

Common access control vulnerabilities include:

- Violation of least privilege or deny by default.
- Bypassing access control checks.
- Elevation of privilege, manipulation or misconfiguration to gain access that would have otherwise been unauthorized.

Consider that 94% of applications were tested for some form of broken access control with the average incidence rate of 3.81%. Most notable Common Weakness Enumerations[2] (CWE) in Broken Access Control include *CWE-200: Exposure of Sensitive Information to an Unauthorized Actor* and *CWE-201: Exposure of Sensitive Information Through Sent Data*, two categories directly related to data exposures.

2: A category system for weaknesses and vulnerabilities.

We propose ways to improve Broken Access Control in Chapter 4 and Chapter 5. Namely, we show how to verify the implementation of a policy corresponds to its specification in Chapter 4, and try to find and eliminate policy redundancies in Chapter 5.

The second on the list, *Cryptographic Failures*, is also very relevant to us. This vulnerability has moved up from third place to second place, and was labeled *Sensitive Data Exposure* in previous editions of the OWASP Top 10 [29]. This category deals with cryptographic failures, which often lead to the exposure of sensitive data.

Common access control vulnerabilities include:

- Data transmitted in clear text.
- Old weak cryptographic algorithms or protocols.
- Deprecated or re-used keys and hash functions.

We propose ways to mitigate those vulnerabilities in Chapter 3.

Data exposures are happening more and more

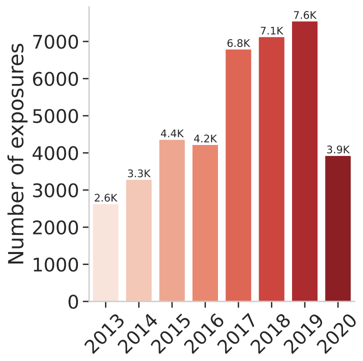


Figure 1.3: Number of exposures reported each year from 2013 to 2020 [14].

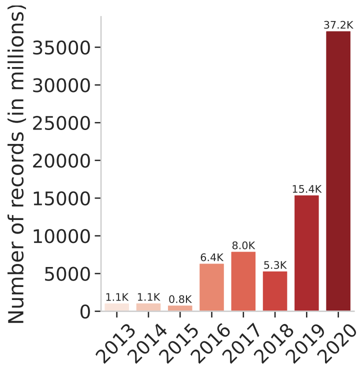


Figure 1.4: Number of records lost each year (in millions) from 2013 to 2020 [14].

We can unfortunately observe that data leaks and breaches are increasingly happening. Both the number of exposures and the number of exposed records have been on an upward trend [13–15] (Figure 1.3 and Figure 1.4). The number of exposed records was over 40 billion in 2021 [13], up from 37.2 billion for 3,932 incidents in 2020 [14] and from 15.4 billion for 7,553 incidents in 2019 [15]. This is a lot more than a decade ago where the number of exposures was at 2,644 and the number of exposed records was at 267 million [16]. Risk Based Security specify in their 2020 report [14] that hacking has been responsible for most incidents, but accidental exposure of data on the Internet (e.g. misconfigured databases, backups, end points, services) has put the most records at risk.

Indeed, the number of data breaches in particular is huge, and impact a wide number of organizations such as government agencies, retail businesses, credit agencies, movie studios and many more [6].

Other sources corroborate those findings. The Privacy Rights Clearinghouse provides access to a dataset containing a chronology of data breaches [18] . They rely on breach information published by government agencies that receive notices directly from breached businesses as a result of regulatory obligation. For each breach, they give the name of the organization, the city, the type of breach, the type of organization and the number of breached records. They also provide a description of each incident.

As the time of writing, this dataset contains almost 10.4 billion breached records coming from more than 9000 data breaches. An analysis of the dataset done in 2016 [30] reveals that the main cause for data breaches was hacking by an outside party or infected by malware. Looking at the type of organization most affected, the MED[3] category has the highest number of data breaches, followed closely by BSO[4] . The analysis also found that the BSR[5] and BSO categories were the main target of hackers, and that GOV[6] breaches were mainly due to unintended disclosure. In the latter, sensitive information was posted publicly, mishandled or sent to the wrong party via publishing online, sending in an email, sending in a mailing or sending via fax.

3: MED - Healthcare, Medical Providers and Medical Insurance Services

4: BSO - Businesses (Other). Businesses besides financial and insurance services and retail.

5: BSR - Businesses (Retail/Merchant including Online Retail).

6: GOV - Government and Military.

[18]: Clearinghouse (2020), *Data Breaches*

Preventing both forms of data exposure is a challenging problem to tackle, since as we have seen, an exposure can occur in multiple circumstances, caused by hacking, misconfigured databases, malicious insiders, etc. Note that those exposures are perceived as huge losses of money for businesses such as the movie industry [17] , and as a loss of user privacy for applications dealing with user data [18].

[17]: Byers et al. (2003), 'Analysis of security vulnerabilities in the movie production and distribution process'

The cloud has a part of responsibility

With more and more businesses using public clouds to process data [31] and deploy their workflows [32] , and with data being frequently moved around, exposures are more likely to happen than ever.

[32]: Mehta et al. (2017), *How Netflix Is Solving Authorization Across Their Cloud*

Indeed, the public cloud service market was valued at $371.4 billion in 2020 and is projected to reach $832.1 billion by 2025 [33]. An estimated 94% of organizations already use a cloud service, and 48% of businesses store classified and important data in the cloud. Data stored in the cloud has reached 44 zettabytes[7] in 2020 and is projected to reach 200+ zettabytes by 2025 [34], the entirety of which will require protection.

7: 44 billion terabytes.

Human error is to blame for cloud data breaches in 88% of cases [33].

1.3 A change of paradigm

Over the years, the approach to creating and delivering software has changed. We have evolved from a monolithic approach to a more modular approach, from local on-premise deployments to the cloud, and this has had an effect on security risks.

From monolithic to modular

The traditional way applications are made involve a monolithic framework. A monolithic application has a tightly coupled codebase, where code is packaged together and makes assumptions about the other parts of the code. Over the years, and with the emergence of the cloud, people have increasingly turned to a modular approach when it was relevant to the use case. Figure 1.5 is a typical representation of a monolithic application.

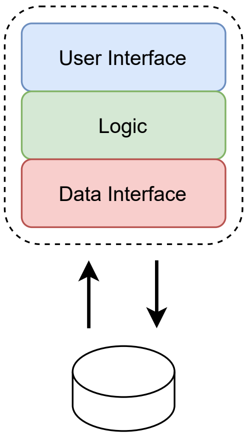


Figure 1.5: Monolithic approach.

With this modular approach, the codebase is loosely coupled, where the code communicates with other parts of the code through standardized interfaces. In this instance, code is more fragmented, usually leading to better readability because it results in smaller single-purpose fragments of code. The codebase is also more maintainable, because bugs can be fixed and features can be added by simply modifying one or more of the modules, as opposed to having to refactor the entire codebase. Modularity makes the codebase more portable, once again because it is separated in independent modules. In addition, it also makes it easier for individuals to work independently on different parts of the code, which can lead to faster development and time to market. Finally, the modular approach helps with the scalability of applications, in particular if one part of the codebase scales differently from the others. The modular approach has recently been incarnated by microservices. Figure 1.6 illustrates this approach.

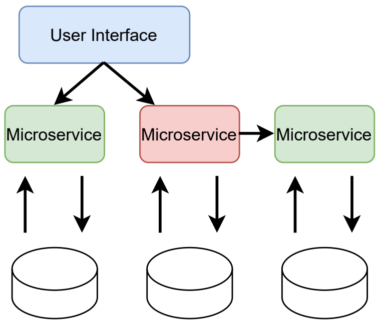


Figure 1.6: Modular approach.

Although arguments can be made for both approaches, with each one having the possibility of being more fitting depending on the application, the fact is that an increasing number of organizations use the modular approach. In fact, a 2020 survey done by O'Reilly found that 28% of respondents had been using microservices for at least three years, and that 61% of the respondents had been using them for a year or more [35]. Further, only 23% of respondents said they were not currently using microservices.

From on-premise to cloud

In the same way, application were traditionally deployed on the premises of its company. In an on-premise deployment, applications are hosted on the servers owned by the company, making them bear the cost of hardware, power consumption and space.

Cloud computing, opposing the traditional on-premise deployment, has grown in popularity over the years. In this case, the company deploys its applications on the servers of a service provider, and only pays the cost for the resources they use. This cuts a lot of costs, as the company no longer needs to maintain its environment.

The zero-trust security model

The fact that nowadays more people use the cloud and that modular approaches are generally preferred has led to a modification of several aspects security-wise.

By deploying their applications to the cloud, businesses relinquish some of the control to the cloud service provider. Their data is usually no longer stored locally, but at one or more remote locations owned by the service provider.

Furthermore, attacks were generally assumed to come from a location external to the deployment via north-south traffic. The traditional way to secure such a deployment against those attacks was to protect it at the border via gateways, firewalls or programmable switches [36] (Figure 1.7). The rise of the cloud and microservices as a paradigm, as well as their increased use in building large, cloud-based enterprise applications [37] has increased the attack surface, meaning protecting the network border is no longer sufficient.

[36]: Jin et al. (2016), 'Understanding security group usage in a public iaas cloud'

[38]: Gilman et al. (2017), *Zero Trust Networks*

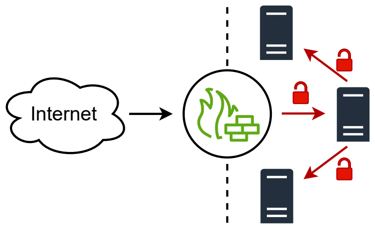


Figure 1.7: Traditional way to secure an application. The deployment is secured at the border. Inside communications are not protected.

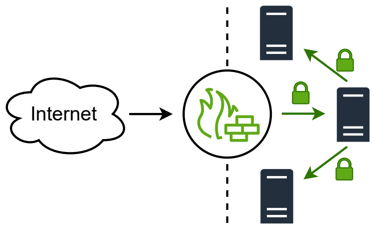


Figure 1.8: Application security under the zero-trust model. The deployment is secured at the border, and inside communications are protected.

To prevent data exposures in such an environment, one also needs to consider exposures coming from inside the system. Those exposures can either be leaks stemming from the way data is processed or breaches caused by a malicious employee. The zero-trust security model [38] , where all traffic flows are required to be authenticated and authorized via fine-grained policies, provides such protection (Figure 1.8). This model embodies the *"Never trust, always verify"* principle. In zero-trust, users need to always be given the least amount of privileges, and the system as a whole needs to be monitored.

1.4 Authentication and authorization

With an increasing number of organizations moving to the cloud and the security concerns this entails, as well as the always present risk of exposures, the situation sure seems dire. However, researchers have come up with ways to prevent or mitigate those risks, namely in the form of authentication and authorization. Authentication is used to verify whether a user is who he claims to be, whereas authorization is used to define what users can access. This section broadly presents both those mechanisms.

Verifying one's identity using authentication

The purpose of authentication is to identify users. Authentication is usually done before authorization, and relies on one (or more) mechanisms. Those authentication solutions can in general be associated to one of three categories. They either rely on (i) something in the possession of the user, (ii) something the user knows, or are (iii) based on an inherent characteristic of the user [39] .

[39]: Barkadehi et al. (2018), 'Authentication systems: A literature review and classification'

For example, ownership-based authentication includes physical keys, Trusted Platform Modules (TPM), Public Key Infrastructures (PKI), certificates and tokens. Knowledge-based authentication includes passwords, graphical passwords and challenge response quizzes. Lastly, inherent-based authentication includes biometrics or even a user's behavioral data.

A good authentication solution needs some specific characteristics in order to be considered useful [40] . Overall, such a solution needs to score well in several dimensions:

[40]: Bonneau et al. (2012), 'The quest to replace passwords: A framework for comparative evaluation of web authentication schemes'

- Security
  The solution needs to resist to attacks.
- Usability
  The solution must be scalable and easy to learn. The solution should provide an easy way to recover credentials. Having nothing to carry is also considered a benefit from the point of view of usability.
- Deployability
  The solution should have a negligible cost per user to implement and should be compatible with the system it is deployed in.

Depending on the specific use case, some additional constraints/requirements might arise. For example, an authentication solution that aims to integrate well with workflows might need to deal with short-lived identities and frequent changes in the userbase. A solution that aims to integrate well with the cloud might need to be especially usable and scalable.

As one can probably tell, there is no perfect solution, each will have it's own advantages and drawbacks in those different dimensions. For instance, using passwords might be less secure that having a hardware token, but the deployability and usability of hardware tokens is much less than that of passwords.

Defining permissions using authorization

The term *authorization* is often used interchangeably with the term *access control*. To be concise, authorization refers to the policy that defines what users can access, whereas access control refers to the mechanisms used in practice to enforce this policy.

Authorization is a principal aspect of security, regulating the interactions taking place in a given system. Authorization controls how users in a system can access resources, answering the question "Who is allowed to do which action on which resource?".

Since the systems to be secured by authorization can be very complex, administrators often rely on policy-based management of authorization. Policies define the desired behavior of a system from a high-level perspective. Hence, this form of management allows to separate the problem of specification, i.e. defining the desired system behavior, from the problem of implementation, that is the enforcement of the desired system behavior. This makes authorization management easier and more flexible. To give an example, a policy specification might consist of an access matrix, a privilege graph or a YAWL document, while a policy implementation might be an access control list, an XACML document or a Rego document.

1.5 Contributions of this thesis

In previous sections, we have drafted a landscape of the current situation. Workflows are used everywhere and by everyone, multi-party workflows in particular being needed whenever there is a collaboration between two or more organizations. Data leaks and breaches are widespread, regular outcomes of the most critical vulnerabilities, and happening continuously more. In addition, the shift to the cloud and modular approaches has increased attack surfaces and brought new security risks that did not exist before.

Considering this situation, with major companies storing their data unencrypted in the cloud to enable their multi-party workflows, there is a strong need for a solution that can enable those workflows securely and prevent exposures.

This thesis therefore focuses on the prevention of data exposures, in workflows in particular. Our goal is to enable secure multi-party workflows and mitigate the risk of data exposures.

The most critical vulnerabilities causing data exposures are *Broken Access Control* and *Cryptographic Failures*. In accordance with this fact, we have structured this thesis around three axes that try to mitigate those vulnerabilities.

In Chapter 3, we propose an infrastructure using microservices to enable secure multi-party workflows and deal with data exposures. We realize a proof of concept of this infrastructure, and deploy it on Google Cloud Platform. The focus of this chapter is more on cryptographic failures, as we give ways to secure data at rest as well as in transport.

- ► *RQ1: How can we construct a leak-free multi-party workflow?*

Chapter 4 and Chapter 5 deal with broken access control. In the former, we took an interest in the verification of deployed security policies. We investigate policy-based management of authorization, and more specifically ways to verify that a policy specification matches its implementation. We develop a tool enabling policy verification with the help of metagraphs, a generalized graph theoretic structure. Implementations are not free of errors, and a verification mechanism can mitigate the risk of having broken access control.

- *RQ2: How do we verify a policy specification corresponds to its deployed implementation?*

In Chapter 5, we deal with policy analysis. In particular, we analyze policy specifications to detect and eliminate potential redundancies. We propose a way to detect redundancies, and develop a tool to do so. By removing clutter from policy specifications, we hope to give policy administrators a clearer view of the policy. In this way, we can help them make less errors when they are handling specifications, and therefore reduce the risk of broken access control.

- *RQ3: How do we verify a policy specification contains no redundancies?*

We summarize our contributions in Table 1.1.

Table 1.1: Summary of contributions.

Chapter	Topic	Publication
Chapter 3	Secure infrastructure	[19, 20]
Chapter 4	Policy verification	[20, 21]
Chapter 5	Policy redundancy elimination	[22]

We also have developed several tools that pertain to each contribution. We summarize the tools we developed in Table 1.2.

Table 1.2: Summary of tools.

Chapter	Tool	Repository
Chapter 3	Proof of Concept	`https://github.com/loicmiller/secure-workflow`
Chapter 4	Policy verification with metagraphs	`https://github.com/loicmiller/policy-verification`
Chapter 4	MGToolkit for Python 3	`https://github.com/loicmiller/mgtoolkit`
Chapter 5	Policy redundancy elimination	`https://github.com/loicmiller/policy-analysis`
Chapter 5	SAT formulation	`https://github.com/loicmiller/fhep-sat-formulation`

In this section, we have given a brief overview of the contents of this document. More details about each contribution can be found in the following outline, or in their corresponding chapter.

1.6 Outline

The remainder of this thesis is split into five chapters.

Workflows, the cloud and authorization

We have briefly presented in the previous sections a landscape of the current situation. In particular, we have introduced numerous concepts, namely workflows, the cloud, and security mechanisms surrounding authorization. In Chapter 2, *Background and Related Works*, we will dive into those concepts and more.

First, we develop the notion of workflows. While the idea of a workflow can evoke different images depending on the reader, we will give workflows a working definition. We then present elements related to workflows, and show how they are modeled by the use of a specification language.

Next, we introduce concepts pertaining to the cloud. Technologies surrounding the cloud have evolved quickly over the years, microservices being one of the last iterations to date. We explain how microservices work from the ground up, starting with the basic container, and then explain how higher layers of the microservice architecture are built on top of them. Namely, we detail how orchestrators and service meshes work. We also touch on related works pertaining to microservices, zero-trust and formal analysis of leaks.

Then, we go deeper into research fields surrounding authorization, specifically ones related to policy. We detail existing archetypes used to specify a policy. We describe how policies go from their initial specification to their implementation, as well as how they are analyzed and verified.

Lastly, we introduce two graph theoretical structures, metagraphs and hypergraphs. Numerous structures have already been used to represent policies. We argue that metagraphs, although underused for this purpose, constitute one of the best ways to model policies. Those structures will be used extensively to model, manage, verify and analyze policies throughout Chapter 4 and Chapter 5.

A secure infrastructure to prevent data exposures

In Chapter 3, *A Secure Infrastructure to Prevent Data Exposures*, we propose an infrastructure to deal with data exposures. This infrastructure makes use of the aforementioned microservices, and provides ways to secure data at rest and in transport.

First, we present our security model. This model is composed of a threat model, where we consider threats from different points of view in the workflow. It is also composed of a trust model, that specifies our trust assumptions when it comes to workflow participants.

Second, we detail the construction of our infrastructure, and present our proof of concept. We describe how each layer of the microservice architecture is used, and what need they fulfill security-wise. We present our proof of concept deployed on Google Cloud Platform (GCP), as well as a basic policy verification mechanism.

Finally, we measure the overhead cost in terms of performance regarding our infrastructure. More specifically, we measure startup times and request delays, showing the overall cost to be manageable.

A way to verify policies using metagraphs

Although we argue the secure infrastructure of Chapter 3 is a good basis to prevent both leaks and breaches, some attack vectors still remain. For some of those, we unfortunately have little control over the situation. Indeed, it is for example still possible to exploit a flaw in a service running on our infrastructure. Since that service can be literally anything, there is not much more we can do to prevent attacks. For some other attack vectors however, there are some additional measures we can take.

Specifically, we can perform some supplementary checks on the policy used in our infrastructure. Recall that broken access control is the most critical vulnerability right now, and that human error causes 88% of cloud data breaches. In that spirit, we propose in Chapter 4, *Verifying Policies Using Metagraphs*, a way to verify that the policies deployed in our infrastructure actually match their initial specification, and that no errors were introduced.

We do this by first showing how workflows can be modeled using metagraphs. We go over the different elements composing workflows, and show how they are translated in their metagraph counterpart.

Then, we explain how those metagraph models are used to verify policies. In particular, we compare the metagraph modeling the specification of the policy to the metagraph modeling the implementation of the policy. Any difference in the metagraphs show that errors are present, be it by human error or by some other way.

Last, we conduct a performance analysis of our verification process, and show the method to be efficient. To obtain general and representative results, we generate random workflows. We provide tools for each of the previously presented steps, from generation to translation to comparison.

Policies, redundancies, and complexity results

In Chapter 4, we devised a way to verify policies. Using this method, we can make sure the implementation of a policy corresponds exactly to its specification. However, if the specification itself has problems, our verification method will fail to capture them. This is what Chapter 5, *Analyzing Policies to Find Redundancies*, is dedicated to.

In this chapter, we tackle the question of redundancy elimination in a policy. We study and improve a currently existing solution. We prove the problem of finding redundancies is NP-hard in the general case.

In accordance with this result, we translate our problem to a SAT formulation but find it is not efficient. Ultimately, we devise a more efficient algorithm using Pascal's triangle to find redundancies. We show our algorithm is more efficient by conducting a performance analysis on the methods detailed in this chapter, and comparing them.

Finally, we conclude with Chapter 6, *Conclusion and Research Directions*. We summarize our propositions, and present the main takeaways from this thesis. We discuss our solution in its different aspects, and examine their limitations. We conclude by discussing research perspectives, and outline possible future works.

1.7 List of contributions

1.7.1 Publications

Journals

- Loïc Miller, Pascal Mérindol, Antoine Gallais, Cristel Pelsser. *Securing Workflows Using Microservices and Metagraphs*. MDPI Special Issue Advances in Communications Software and Services. December 2021.

Journal preprints

- Reynaldo Gil Pons, Max Ward and Loïc Miller. *Finding (s,d)-Hypernetworks in F-Hypergraphs is NP-Hard.* arXiv preprint 2201.04799. January 2022.

International conferences

- Loïc Miller, Pascal Mérindol, Antoine Gallais, Cristel Pelsser. *Towards Secure and Leak-Free Workflows Using Microservice Isolation*. International Conference on High-Performance Switching and Routing (HPSR). June 2021.
- Loïc Miller, Pascal Mérindol, Antoine Gallais, Cristel Pelsser. *Verification of Cloud Security Policies*. International Conference on High-Performance Switching and Routing (HPSR). June 2021.

National conferences

- Loïc Miller, Pascal Mérindol, Antoine Gallais, Cristel Pelsser. *Protection contre les fuites de données: un environnement micro-services sécurisé*. CORES 2021–6ème Rencontres Francophones sur la Conception de Protocoles, l'Évaluation de Performance et l'Expérimentation des Réseaux de Communication. September 2021.
- Loïc Miller, Pascal Mérindol, Antoine Gallais, Cristel Pelsser. *De l'Utilisation des Métagraphes pour la Vérification de Politiques de Sécurité*. ALGOTEL 2021—23èmes Rencontres Francophones sur les Aspects Algorithmiques des Télécommunications. September 2021.

1.7.2 Tools

All tools listed are licensed under the MIT license.

A Secure Infrastructure to Prevent Data Exposures

- Proof of Concept - `https://github.com/loicmiller/secure-workflow`.

Verifying Policies Using Metagraphs

- Policy verification with metagraphs - `https://github.com/loicmiller/policy-verification`.
- MGToolkit for Python 3 - `https://github.com/loicmiller/mgtoolkit`.

Analyzing Policies to Find Redundancies

- Policy redundancy elimination - `https://github.com/loicmiller/policy-analysis`.
- SAT formulation - `https://github.com/loicmiller/fhep-sat-formulation`.

Background and Related Works | 2

In the introduction, we have broadly addressed the topics of workflows, the cloud, and security mechanisms surrounding authorization. In Chapter 2, we dive more into those concepts as their understanding is required for the following chapters.

We first develop in Section 2.1 the notion of workflows. We give them a working definition, and introduce Yet Another Workflow Language (YAWL), a workflow language. We introduce elements of YAWL and show how they are used.

Next, we introduce in Section 2.2 concepts pertaining to the cloud, namely microservices. We explain level by level the different components of the microservice architecture. We then present related works on this theme and close to our contributions.

In Section 2.3, we go deeper into research fields surrounding authorization, specifically ones related to policy. We detail the main archetypes when it comes to authorization, and present related works in this field.

Lastly, we introduce in Section 2.4 two graph theoretical structures, metagraphs and hypergraphs. We define those constructs as well as concepts pertaining to them, and present related works. Those structures will be used extensively in Chapter 4 and Chapter 5.

2.1 Workflows

2.1.1 Workflows, informally

The idea of a workflow can evoke different images depending on the reader. Informally speaking, a workflow is a sequence of tasks that processes data [41]. Some people will think of workflows where tasks are predictable and repetitive. These are called process workflows. An example of those are workflows handling customer orders. In this case, every order follows the same workflow.

Others might think of case workflows, where the workflow depends on dynamic data. A simple example are support and ticket systems, where some inquiry is needed to process the data. Here, two tickets might follow different workflows depending on their contents.

Finally, some people will think of project workflows, which are similar to process workflows. The difference here is that project workflows are only useful for the current project, and will probably not be reused for another project. A typical example of this can be found in software development, where the workflow might be different based on what the end goal is.

We define a workflow as a sequence of tasks to be performed by a set of independent actors. The owner of the data (i.e., the instigator of the workflow) interacts with contractors to realize such a sequence. Both the owner and the contractor have agents processing the data, where agents can represent an employee or an automatic service.

Throughout this document, we use a recurring example pertaining to the movie industry. Consider a simple workflow, where an owner in the post-production stage of making a movie employs other companies to edit the video and audio components [17] .

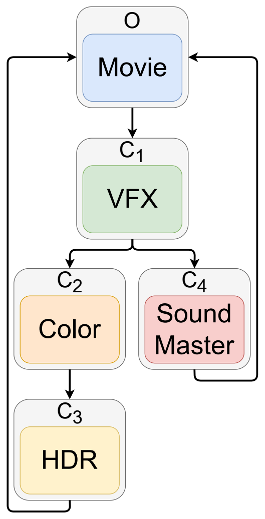


Figure 2.1: Movie workflow example. Arrows model the specified workflow, and thus represent the communication flow. The owner (O) sends its data to the first contractor (C_1) for special effects processing. C_1 then sends the modified data along the workflow, for color (C_2), HDR (C_3) and sound (C_4) processing. The resulting data is then sent back to the owner.

[17]: Byers et al. (2003), 'Analysis of security vulnerabilities in the movie production and distribution process'

Figure 2.1 is a representation of such a workflow. The owner (O) first sends its data to the company responsible for special effects (C_1). C_1 applies special effects to the movie sequences the owner sent him, and then sends the result to the company responsible for coloring (C_2) as well as another for sound mastering (C_4). C_2 then ships its result to the agent in charge of High Dynamic Range (HDR) (C_3). Finally, both C_3 and C_4 sends their output back to the owner, which puts the data together to obtain the final product.

2.1.2 Workflows, formally

To formally represent a workflow, we need a way to specify them. The area of Business Process Management (BPM) has received a lot of attention over the years, and several languages that can represent processes and workflows have been developed.

Initially, many BPM tools supported their own language [42]. There has been attempts to define a standard, the XML Process Definition Language [43] (XPDL) being one of the first. Defined by the Workflow Management Coalition (WfMC), the language's minimalist approach and the fact that even basic constructs could be ambiguous and interpreted in different ways ultimately made the language irrelevant [42]. 2003 saw the apparition of the Business Process Execution Language [44] (BPEL). Developed as a combination of Microsoft's XLANG and IBM's Web Services Flow Language (WSFL), the language was an improvement over previous approaches [42] . Unfortunately, having no graphical representation and providing no support for the involvement of human participants, it was quickly abandoned to the profit of Business Process Model & Notation [45] (BPMN). With its first specification coming out in 2006, BPMN provided a graphical representation and more support of workflow patterns [42]. However, BPMN still lacked some support for the involvement of human participants, and the interpretation of some of its concepts could vary. Nonetheless, it was partly adopted, and is still in use to this day.

[42]: Ter Hofstede et al. (2009), *Modern Business Process Automation: YAWL and its support environment*

[45]: Business Process Model (2011), 'Notation (bpmn) version 2.0'

Another approach lies in Unified Modeling Language [46] (UML) with their activity diagrams. Two versions are worth mentioning, the 1.4 version inspired by statecharts, and the 2.0 version inspired by Petri nets [42]. Despite being modeled after Petri nets, some Petri net concepts could not be mapped to UML 2.0. UML activity diagrams were not intended for direct execution, and with no simple and clear semantics as well as no formalization, BPMN was still preferred to specify business processes [42].

After revisiting Petri nets, it was concluded that although many patterns could be expressed in a simple way, some workflow patterns were not easily modeled with them [42]. This finding is what led to the development of Yet Another Workflow Language [47] (YAWL). YAWL extends Petri nets, by adding formalisms to deal with patterns like cancellation. When compared with previous languages, YAWL provides stronger support for workflow patterns, making it easier to specify complex workflows [48]. YAWL also has formal semantics, meaning constructs are not subject to interpretation, and support for dynamic workflows, granting more flexibility in specification. As opposed to the other languages, YAWL models are executable, and YAWL provides a graphical representation. All those qualities make YAWL the overall better option when it comes to specifying workflows.

[47]: Van Der Aalst et al. (2005), 'YAWL: yet another workflow language'

Distinction between processes and workflows

Although we have up until now used the term workflow extensively to describe both processes and workflows, a distinction is to be made between them. A process may contain information elements which are not yet evaluated, whereas a workflow is an instantiation of a process for a set of particular values. For example, one can specify a process for handling customer orders, with many elements that are not evaluated (e.g. the contents of the shopping cart). In one particular instance of this process where a customer actually sets values to those elements, the process is not a process anymore, it is a workflow. As such, a process can result in multiple workflows.

A simple case

A process specification in YAWL represents a sequence of tasks, and has a start and an end. Basic tasks are called atomic tasks, and represent simple actions. The start and end states of the process are different from atomic tasks, as they model states in the process. As such, these states are not modeled by atomic tasks, but rather by conditions. YAWL also uses conditions (besides the start and end), which represent a state the workflow is in after finishing a task, but before starting a new one. Those conditions are useful when users make some decisions on their own and the workflow system cannot pre-determine their choices. Figure 2.2 displays a simple process with one task representing the processing of an order, one condition which asks if the task was done correctly, as well as the start and end conditions.

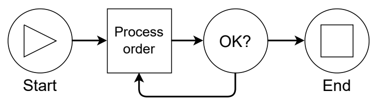


Figure 2.2: Simple process with one task, and a condition which asks if the task was done correctly. The process also exhibits start and end conditions.

YAWL nets

More generally, a specification in YAWL is a set of extended workflow nets which form a hierarchy [47]. Each of these nets represents a process or a sub-process. The net at the top of the hierarchy is called the root net. Tasks in a process can be either atomic or composite, where composite tasks represent another net, but at one lower level in the hierarchy. As said above, each net has one input condition and one output condition, which represent the start and end points of the process. Figure 2.3 displays the same process as in Figure 2.2, but with a composite task, that is another net containing more detailed tasks.

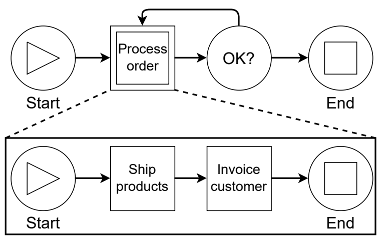


Figure 2.3: Same process, but more detailed with a composite task.

This separation in nets is useful when the process becomes too complex to manage, and can be broken down in smaller pieces. For example, a process for an online store might comprise a `checkout` task, but this task can be comprised of multiple tasks such as `choose shipping` or `choose payment mode`. To model this example in YAWL, the `checkout` task would be the composite task in the root net, while the other tasks are part of the `checkout` net at one lower level in the hierarchy.

Operators

There is the possibility to use operators, namely `AND`, `OR`, and `XOR`, which control the flow of information between tasks of the process. Each of these operators have a split and a join variation, which indicate the behavior of the operator.

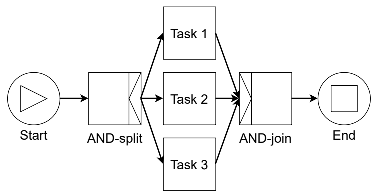


Figure 2.4: Process with an `AND-split` and an `AND-join`.

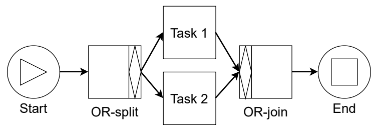


Figure 2.5: Process with an `OR-split` and an `OR-join`.

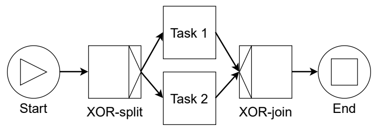


Figure 2.6: Process with a `XOR-split` and a `XOR-join`.

- An `AND-split` indicates the workflow executes all branches of the split, whereas an `AND-join` indicates the tasks reaching the join must all be completed for the next task to be executed. Figure 2.4 represents both variations of this operator.
- An `OR-split` indicates the workflow executes one or more branches of the split, whereas an `OR-join` indicates at least one of the tasks reaching the join must be completed for the next task to be executed. Figure 2.5 represents both variations of this operator.
- A `XOR-split` indicates the workflow executes exactly one branch of the split, whereas a `XOR-join` indicates exactly one of the tasks reaching the join must all be completed for the next task to be executed. Figure 2.6 represents both variations of this operator.

A real-world example

[49]: Business Process Management (BPM) Group (2010), *YAWL4Film*

All those elements can be combined together to realize a process. For example, let us consider the film production process represented in Fig. 2.7, representing the chain of information processing realized along the shooting of the movie [49] . The film production process covers the entire period dealing with the shooting of the movie. While the shooting is taking place during the day, a designated crew collects the information associated with the shooting via production forms (e.g. camera sheets, sound sheets). All those forms are then gathered to produce the daily progress report (DPR), which summarizes the shooting information to keep track of progress and expenses. In addition to the generation of the DPR, the crew monitors requirements for the next shooting days to create the call sheet for the next day, which contains all the information concerning logistics and necessities. A call sheet containing all the information concerning logistics and necessities is also created. This process is carried out on a daily basis.

This process taken from the YAWL foundation website was realized in collaboration with the Australian Film Television and Radio School (AFTRS), with the help of domain experts. As such, one can see how YAWL elements are used in a realistic context. For example, a condition is used with the `Fill Out Sound Sheets` task, indicating the need for this information to be checked by the production manager before proceeding to task `Create DPR` [49].

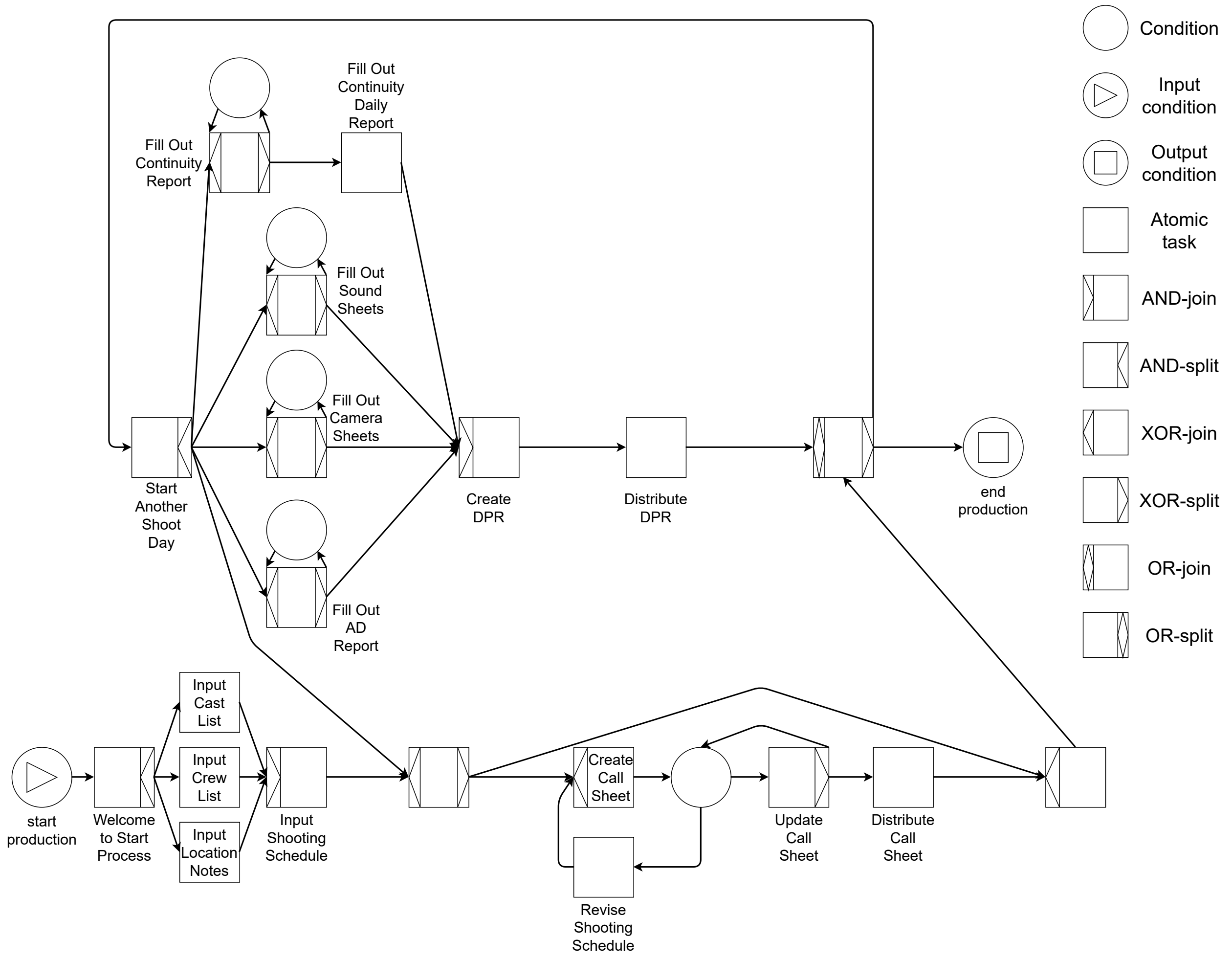


Figure 2.7: Film production process represented in YAWL. This case study represents the chain of information processing realized along the shooting of the movie. While the shooting is taking place during the day, a designated crew collects the information associated with the shooting via production forms, which are used to produce the daily progress report. A call sheet containing all the information concerning logistics and necessities is also created.

2.2 Microservices

Technologies surrounding the cloud have evolved quickly over the years, microservices being one of the last iterations to date. In recent years, microservices have become the de-facto standard way of developing cloud-native apps [37] . Among the different ways microservices can be deployed, service meshes have been characterized as the latest evolution in software service design, development and delivery [50] .

[37]: Chandramouli et al. (2020), *Building Secure Microservices-based Applications Using Service-Mesh Architecture*

[50]: Jamshidi et al. (2018), 'Microservices: The journey so far and challenges ahead'

2.2.1 Components

Here, we show how microservices work from the ground up, starting with the basic container, and then explaining how higher layers of the microservice architecture are built on top of them. We see how service meshes are built on top of orchestrators, which are themselves built on top of containers. We will also detail the use of policy engines in service meshes.

Containers

Containers can be described as a standard unit of software packaging application code, called a service, and its dependencies in an isolated environment. They grant portability and a standardized environment that enables easy adoption and scaling. Some examples of container technology include Docker [23] and Linux containers (LXC, LXD) [51].

[23]: Docker (2022), *Docker*

In reality, a container corresponds to several defined namespaces which gives the service its own view of certain resources. There are six namespaces in total which contribute to the isolation, flexibility and standardization of the container (Figure 2.8):

- mnt: The mnt namespace changes the processes' view of the filesystem.
- uts: The uts namespace isolates the hostname.
- pid: The pid namespace isolates PIDs.
- user: The user namespace isolates user IDs.
- ipc: The ipc namespace isolates IPC System V and SHM.
- net: The net namespace isolates networking.

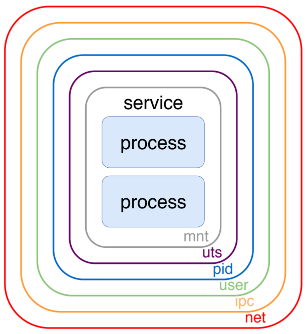


Figure 2.8: A container and its namespaces. The namespaces give the service its own view of certain resources.

Orchestrators

An orchestrator is a system used to automate the management of containers and their life cycles. The orchestrator does this by interacting with workers running on physical machines, which control groups of containers called pods. Some examples of orchestrators include Kubernetes [24] and Nomad [52].

[24]: Kubernetes (2022), *Kubernetes*

Figure 2.9 represents an orchestrator and its workers. Each worker has its own agent which instantiates the pods on the worker. The orchestrator will instantiate pods according to the number of replicas needed as well as a strategy, like load balancing services on workers.

In reality, a pod is also composed of several namespaces, which are the same as the ones for the containers. The difference with the pod is that the services existing in the pod share some of the namespaces.

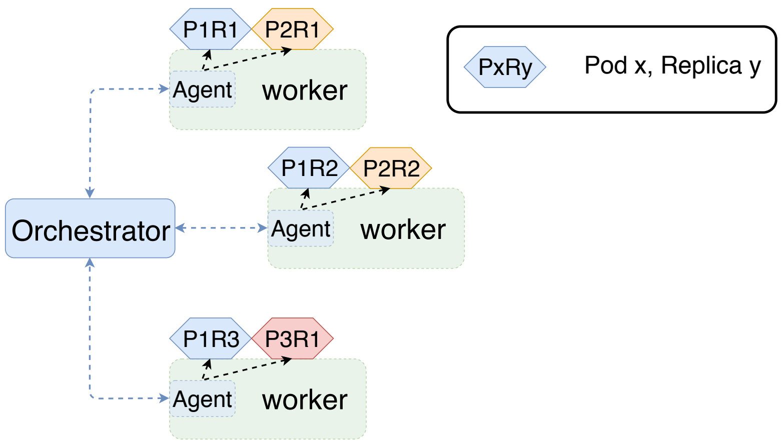


Figure 2.9: An orchestrator and its workers. Each worker has its own agent which instantiates the pods on the worker.

Figure 2.10 represents a pod and its services. As you can see, each service has its own view of the filesystem (mnt) as well as its own view of the hostname, but shares the view of all the other namespaces with the other services inside the pod. Namely, they share the same view of the network namespace. This has many benefits, including but not limited to isolation, management of the pod lifecycle, management of communications.

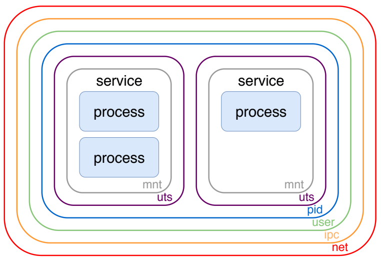


Figure 2.10: A Pod. Compared to containers, services existing in the pod share some of the namespaces.

Service meshes

A service mesh is a dedicated infrastructure layer used to automate the communication, security and monitoring of containerized environments. The deployment of distributed services using only orchestrators can get complex as they grow in size. Such systems get harder to manage, and some aspects like the communication between services and security can become problematic. A service mesh addresses those problems, by providing a simple way to secure and specify communications between services.

There are multiple ways to deploy a service mesh. One can embed the components in application code, couple them to the application code by implementing them as libraries or implement them as service proxies independent of the application code [37]. In the service proxy version, the service mesh controller configures proxies for each environment, which are added as a sidecar to the pod or embedded directly into the services. Those proxies act as intermediaries between services to secure and control their interactions. In other words, proxies intercept ingoing and outgoing traffic to their associated agent. The service mesh controller also provides the proxies with identities for services as well as a key pair for them to interact securely via mutual TLS authentication (`mTLS`). `mTLS` provides authentication and encryption. Some examples of service meshes include Istio [25] and linkerd 2.0 [53].

[25]: Istio (2020), *Istio*

Figure 2.11 represents a service mesh. A proxy is added to each pod as a sidecar. Those proxies are then managed by a centralized controller, which tells the proxies how to communicate with other services via their respective proxies.

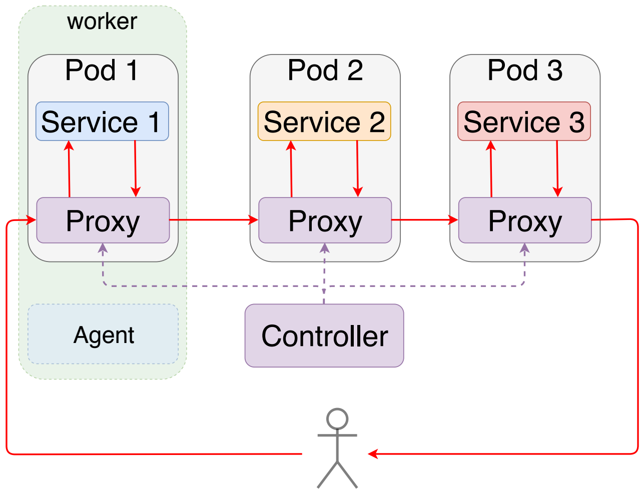


Figure 2.11: A service mesh. A central controller manages proxy sidecars in each pod. Those proxies are used as intermediaries for communication.

Policy engines

A policy engine is a system answering policy-related queries according to a policy configuration.

In the service mesh, every service can communicate with all the other services by default. Adding a policy engine as a sidecar in each pod can enforce a policy on communications. Each time a sidecar proxy intercepts a request from its associated service, it checks with its policy sidecar if the request is authorized or not. If the request is authorized, the request is forwarded accordingly. If the request is not authorized, the proxy returns an error message to the service.

Rules can be added to the policy to constrain the communications a service can make. Additional rules can be used to enforce some other constraints based on context, like a time period during which a given communication is authorized to occur. A policy engine example is Open Policy Agent (OPA) [27] .

[27]: Open Policy Agent (2022), *Open Policy Agent*

2.2.2 Related works

Microservices

Existing works provide guidance on overall security requirements and strategies for microservices [54] , as well as guidance on more specific microservice components such as service mesh [37, 55] or containers [56, 57]. Chandramouli [54] provides guidance on security strategies for implementing core features of microservices, as well as countermeasures for microservice-specific threats. El Malki and Zdun [55] provide guidance on service mesh-based microservice architecture decisions by studying existing practices. Chandramouli and Butcher [37] give recommendations for the deployment of service mesh components in the proxy sidecar version of the microservice architecture.

[54]: Chandramouli (2019), *Security Strategies for Microservices-based Application Systems*

[55]: El Malki et al. (2019), 'Guiding Architectural Decision Making on Service Mesh Based Microservice Architectures'

Souppaya *et al.* [56] provide a listing of major security risks associated with the use of containers, as well as recommendations to counter those risks. Chandramouli [57] then analyzed different security solutions for containers, to see if they were in accordance with recommendations of [56].

Note that those guidelines and recommendations are not specific to workflows, but general good practices for microservices and microservice components. In Chapter 3, we follow the guidelines and recommendations presented in these works. Contrary to those works, we propose a complete infrastructure, accompanied by a real-world deployment, as well as both a security and a performance evaluation of this deployment.

Zero-trust and security

Weever *et al.* [58] investigate operational control requirements for zero-trust network security, and then implement zero-trust security in a microservice environment to protect and regulate traffic between microservices. They focus on implementing deep visibility in the service

[58]: Weever et al. (2020), 'Zero Trust Network Security Model in containerized environments'

mesh, and do not propose a security or a performance evaluation. Hussain *et al.* [59] propose and implement a security framework for the creation of a secure API service mesh using Istio and Kubernetes. They then use a machine learning-based model to automatically associate new APIs to already existing categories of service mesh. Contrary to our work, they use a central enterprise authorization server, in opposition to policy sidecars. Zaheer *et al.* [60] propose eZTrust, a policy-driven parameterization access control system for containerized microservice environments. They leverage eBPF to apply per-packet tagging depending on the security context, and then use those tags to enforce policy, in opposition to our enforcement of policy which relies on policy sidecars local to the services.

[60]: Zaheer et al. (2019), 'eZTrust: Network-Independent Zero-Trust Perimeterization for Microservices'

Formal analysis

On the side of formal analysis of data leaks in workflows, Accorsi and Wonnemann [61] proposed a framework for the automated detection of leaks based on static flow analysis by transforming workflows into Petri nets. Khan *et al.* [62] propose a synthesis on data breach risks and resolutions, identifying and classifying data breaches, locus and impact, and then giving guidance on prevention, containment and recovery. Some papers propose data leak protection, by screening data and comparing fingerprints [63–69]. Segarra *et al.* [70] propose an architecture to securely stream medical data using Trusted Execution Environments, while Zuo et al. investigate data leakage in mobile applications interaction with the cloud [71]. Liu *et al.* [72] and Bosu *et al.*[73] focus instead on leaks in Android apps.

[61]: Accorsi et al. (2011), 'Strong non-leak guarantees for workflow models'

[70]: Segarra et al. (2019), 'Using trusted execution environments for secure stream processing of medical data'

2.3 Policy analysis, refinement and verification

2.3.1 Policy archetypes

There are multiple ways to specify a policy. Over the years and according to their requirements, researchers have come up with multiple archetypes for authorization models. Those archetypes express different views on how authorization should be done, and were designed with specific use cases in mind. There are four main archetypes when it comes to authorization models.

Mandatory Access Control [74, 75] (MAC) is a type of access control in which each user is granted a clearance level, and each resource needs a certain clearance in order to be accessed. Historically, it has been used to protect access to classified information in the United States. Figure 2.12 represents a MAC system. One user has top secret clearance, and can as such access both top secret and unclassified information. The other user only has confidential clearance, and can

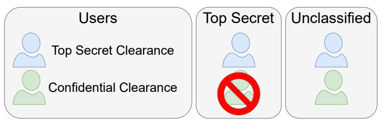


Figure 2.12: Mandatory Access Control.

Discretionary Access Control [76] (DAC) is most associated with Linux systems. Here, each user in the system can grant the permissions he desires to other users or groups, but only for the resources he owns. Figure 2.13 represents a DAC system.

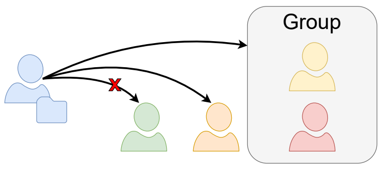


Figure 2.13: Discretionary Access Control.

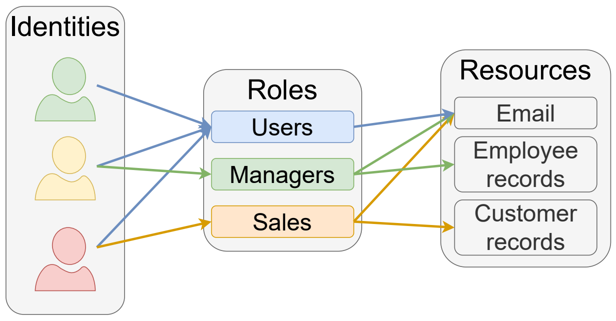


Figure 2.14: Role-Based Access Control.

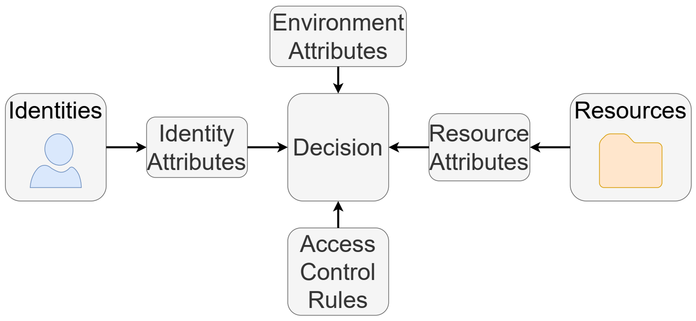


Figure 2.15: Attribute-Based Access Control.

Role-Based Access Control [77, 78] (RBAC) works with identities, roles and resources. Each user (identity) is associated with one or more roles, and each role is then granted permissions. Figure 2.14 represents an RBAC system.

Finally, Attribute-Based Access Control [79] (ABAC) works by assigning attributes to users, resources and other environmental attributes. Access Control rules then decide based on the attributes if a user has access to a resource. Figure 2.15 represents an ABAC system.

A lot of other models exist, including hybrid models [80, 81].

2.3.2 Related works

Along with archetypes which help us find how to specify a policy, there exists several works on policy analysis, refinement and verification in the literature.

Policy analysis

Policy analysis deals with *policy evaluation* and *anomaly analysis*.

In *anomaly analysis*, checking for errors like incorrect policy specifications, conflicts and sub-optimizations affecting either a single policy or a set of policies [82] is the primary research topic. Works in this area use different techniques to achieve this goal, such as model checking [83, 84], binary decision diagrams [85] , graph theory [86] , Deterministic Finite State Automata (DFSA) [87], First Order Logic (FOL) [88] , geometrical models [89], answer set programming [90], petri nets [91] and metagraphs [28].

[82]: Valenza et al. (2017), 'Classification and analysis of communication protection policy anomalies'

[85]: Hu et al. (2013), 'Discovery and resolution of anomalies in web access control policies'

[86]: Koch et al. (2002), 'Conflict detection and resolution in access control policy specifications'

[88]: Cheminod et al. (2018), 'Toward attribute-based access control policy in industrial networked systems'

Policy evaluation instead deals with checking whether a request is satisfied by a set of policies. It is typically used to verify the effective impact of modifying a policy. Works that deal with analyzing the impact of changes in a policy usually model those policies and then analyze the obtained representation for effective impact [92, 93].

Policy refinement

Policy refinement handles the translation from high-level policies into low-level configurations [94] , and has relatively not attracted much efforts. The concept of refinement was introduced by Bandara *et al.* [95] , Rubio-Loyola *et al.* [96] and Craven *et al.* [97] . More recently, works in software-defined networking and network function virtualization have had an interest in policy refinement [98–100].

[94]: Moffett et al. (1993), 'Policy hierarchies for distributed systems management'

[95]: Bandara et al. (2004), 'A goal-based approach to policy refinement'

[96]: Rubio-Loyola et al. (2006), 'A methodological approach toward the refinement problem in policy-based management systems'

[97]: Craven et al. (2010), 'Decomposition techniques for policy refinement'

Policy verification

Finally, policy verification deals with checking whether a policy is correctly enforced in a system. There exists only few works on policy verification when compared to the large body of work dealing with policy analysis. Hughes and Bultan [101] as well as Bera *et al.* [102] propose automatic verification of access control policies against a set of properties. In those papers, verification is achieved by translating the properties into a boolean satisfiability problem and using a SAT solver.

In Chapter 4, and in opposition to those works, we use metagraphs to perform the verification of policies. We handle workflow policies in particular, and show how they can be modeled with metagraphs.

2.4 Metagraphs and Hypergraphs

We introduce two graph theoretical structures, metagraphs and hypergraphs. Numerous structures have already been used to represent policies. We argue that metagraphs, although underused for this purpose, constitute one of the best to model policies. Those structures will be used extensively to model, manage, verify and analyze policies throughout Chapter 4 and Chapter 5.

2.4.1 Definitions - Metagraphs

A metagraph is a generalized graph theoretic structure, which can be defined as a collection of directed set-to-set mappings [103] . Each set in the metagraph is a vertex, and directed edges represent the relationship between sets. More formally, a metagraph can be defined as follows:

[103]: Basu et al. (2007), *Metagraphs and their applications*

Definition 2.4.1 (Metagraph) *A metagraph $S = \langle X, E \rangle$ is a graphical construct specified by a generating set X and a set of edges E defined on the generating set. A generating set is a set of elements $X = \{x_1, x_2, ..., x_n\}$, which represent variables of interest. An edge e is a pair $e = \langle V_e, W_e \rangle \in E$ consisting of two sets, an invertex $V_e \subset X$ and an outvertex $W_e \subset X$.*

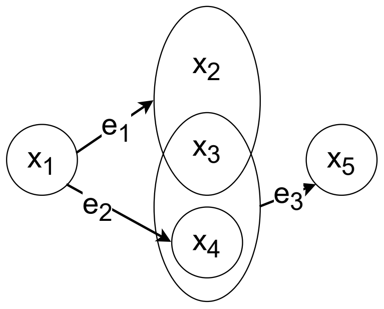


Figure 2.16: A simple metagraph. The generating set is $X = \{x_1, x_2, ..., x_5\}$ and the set of edges is $E = \{e_1, e_2, e_3\}$.

Figure 2.16 depicts a metagraph. In this example, the generating set is $X = \{x_1, x_2, \ldots, x_5\}$ and the set of edges is $E = \{e_1, e_2, e_3\}$. The edge e_1 can be described by the pair $\langle\{x_1\}, \{x_2, x_3\}\rangle$, which are respectively the invertex and outvertex of the edge.

A sequence of edges $\langle e_1, e_2, \ldots, e_n\rangle$ from an element x to an element y where $x \in invertex(e_1)$, $y \in outvertex(e_n)$ and for all e_i, $i = 1, \ldots, n-1$, $outvertex(e_i) \cap invertex(e_{i+1}) \neq \emptyset$ is defined as a simple path. This notion does not describe all the connectivity properties existing in a metagraph. For example, in Figure 2.16, there are two simple paths from $\{x_1\}$ to $\{x_5\}$, $\langle e_1, e_3\rangle$ and $\langle e_2, e_3\rangle$. However, neither of them can calculate x_5 as they respectively do not reach either x_3 or x_4, which are both necessary to calculate x_5. Using the set consisting of all three edges $\langle e_1, e_2, e_3\rangle$ is necessary (and sufficient) to calculate x_5, but it is not a simple path: there does not exist a simple sequence of edges consisting of these three. Such a set of edges $\langle e_1, e_2, e_3\rangle$ is called a metapath [103], and is defined as follows:

Definition 2.4.2 (Metapath) *Given a metagraph $S = \langle X, E\rangle$, a metapath $M(B, C)$ from a source $B \subset X$ to a target $C \subset X$ is a set of edges $E' \subseteq E$ such that every $e \in E'$ is on a simple path from some element in B to some element in C. Moreover, by construction, we have:*

(i) $\forall e = \langle V_e, W_e\rangle \in E', \bigcup_e V_e \setminus \bigcup_e W_e \subseteq B$, i.e., the source should include all pure inputs on the metapath;
(ii) $C \subseteq \bigcup_e W_e$, i.e. the target is included in (a subset of) the union of outvertices.

Reachability between the source and target sets in a metagraph is defined by the existence of (valid) metapaths between the two — in particular verifying the condition on the source that is sufficient by itself to reach the target.

Conditional metagraphs

Metagraphs can be augmented by propositions, i.e. statements that can either be true or false. We call metagraphs without propositions simple metagraphs, and refer to metagraphs with propositions as conditional metagraphs. A proposition attached to an edge can equally be represented as being part of the invertex of the edge, and it must be true in order for the edge to be used in a metapath. Each edge may contain zero or more propositions and each proposition may be used in multiple edges.

Definition 2.4.3 (Conditional Metagraph) *A conditional metagraph is a metagraph $S = \langle X, E\rangle$ with $X = X_p \cup X_v$ where X_p is a set of propositions and X_v is a set of variables. For each edge, at least one of the (in or out) vertices is not empty, i.e., $\forall e \in E, V_e \cup W_e \neq \emptyset$. Second, the invertex and outvertex of each edge must be disjoint. Finally, if the outvertex contains a proposition, it contains only it and does not contain other elements (in X), i.e., with $X_v \cap X_p = \emptyset$, we have $\forall p \in X_p, \forall e \in E$, if $p \in W_e$, then $W_e = \{p\}$.*

Figure 2.17 depicts a conditional metagraph. Here, we show how they can be used by representing the necessary tasks for employees to perform a bank transfer. Employees (u_1, u_2) and tasks ($create\_form$, $fill\_form$,

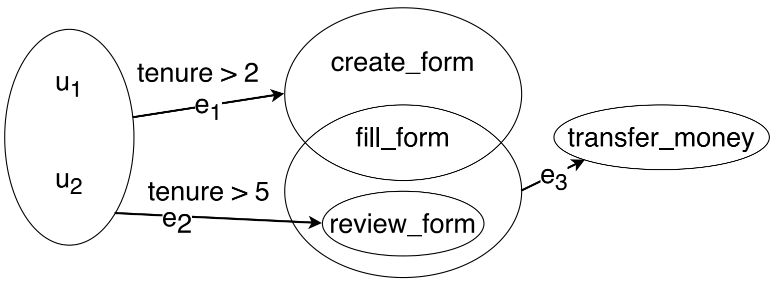


Figure 2.17: A simple example of conditional metagraph to model the following question: what are the necessary tasks for employees to perform a bank transfer? The edge e_3 models the fact that both $fill\_form$ and $review\_form$ need to be fulfilled to perform the last operation $transfer\_money$.

$review\_form$, $transfer\_money$) are put into relation by the edges (e_1, e_2, e_3) between sets of elements. Each edge contains an arbitrary number of propositions (e.g. $tenure > 2$ attached to e_1). If a proposition is set on an edge, it must be true to consider its usage.

Both employees can $create\_form$ and $fill\_form$ via e_1 if they have more than two years of experience, and $review\_form$ if they have more than five years of experience via e_2. The edge e_3 models the fact that both $fill\_form$ and $review\_form$ need to be fulfilled to perform the final operation, $transfer\_money$. Overall, the set of employees $\{u_1, u_2\}$ can perform $\{transfer\_money\}$ if all three edges of the metagraph are used. Note that $\{e_1, e_2, e_3\}$ is not a simple path, but a metapath.

Metagraph properties

Some properties and operations defined on a metagraph are helpful to analyze policies in terms of reachability, redundancy and consistency. For example, metagraphs have a property called dominance which can be used to determine redundant components (edges or elements) [28] .

[28]: Ranathunga et al. (2020), 'Verifiable Policy-Defined Networking using Metagraphs'

Once identified, those components can be safely removed from the policies. A metapath is *input-dominant* if no proper subset of its source is also a metapath to its target, *edge-dominant* if no proper subset of its edges is also a metapath to its target, and *dominant* if it is both input-dominant and edge-dominant [103].

Definition 2.4.4 (Input dominance) *Given a metagraph $S = \langle X, E \rangle$, for any two sets of elements $B, C \subset X$, a metapath $M(B, C)$ is said to be input-dominant if there does not exist any metapath $M'(B', C)$ with $B' \subset B$.*

Definition 2.4.5 (Edge dominance) *Given a metagraph $S = \langle X, E \rangle$, for any two sets of elements $B, C \subset X$, a metapath $M(B, C) = E'$ is said to be edge-dominant if there does not exist any proper subset of edges from E' forming a metapath from B to C.*

Non-dominant metapaths indicate redundancies in a metagraph, as well as in the underlying policies. Considering the example in Fig. 2.18, the metapath $M_1(\{u_1, u_2\}, \{transfer\_money\}) = \{e'_1, e'_2, e_3\}$ is not input-dominant. $M_2(\{u_1\}, \{transfer\_money\}) = \{e_1, e_2, e_3\}$ is indeed a metapath with $\{u_1\}$ a proper subset of $\{u_1, u_2\}$. Similarly, the set of edges $\{e_1, e_2,$

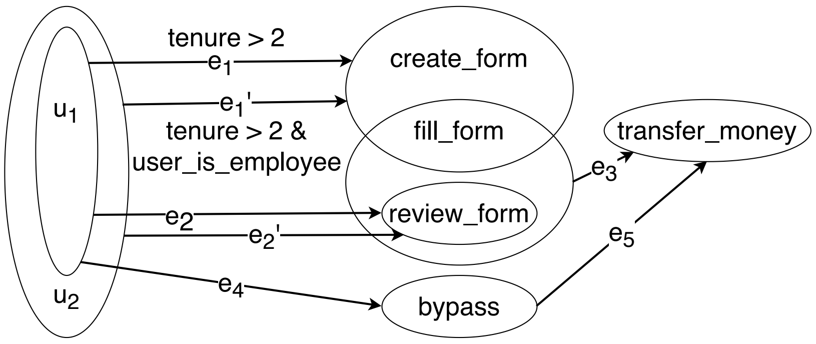


Figure 2.18: This conditional metagraph illustrates the notion of dominance among metapaths. The metapath $M_1(\{u_1, u_2\}, \{transfer\_money\}) = \{e_1', e_2', e_3\}$ is not input-dominant because $M_2(\{u_1\}, \{transfer\_money\}) = \{e_1, e_2, e_3\}$ is also a metapath with $\{u_1\}$ a proper subset of $\{u_1, u_2\}$. Similarly, $M_3(\{u_1\}, \{transfer\_money\}) = \{e_1, e_2, e_3, e_4, e_5\}$ is not an edge-dominant metapath because $M_4(\{u_1\}, \{transfer\_money\}) = \{e_1, e_2, e_4\}$ exists, with $\{e_1, e_2, e_3\} \subset \{e_1, e_2, e_3, e_4, e_5\}$.

$e_3, e_4, e_5\}$ does not form an edge-dominant metapath M_3 for $B = \{u_1\}, C = \{transfer\_money\}$ because one of the dominant metapaths for this couple is $M_4(\{u_1\}, \{transfer\_money\}) = \{e_1, e_2, e_3\} \subset \{e_1, e_2, e_3, e_4, e_5\}$.

2.4.2 Definitions - Hypergraphs

Hypergraphs are similar to metagraphs. They are also a generalization of conventional graphs where sets of elements are connected by a single hyperedge. We are concerned with directed hypergraphs and will thus refer in the following to directed hypergraphs simply as hypergraphs.

Like with metagraphs, each edge in a hypergraph is a directed set-to-set mapping, where the source set is called the *tail* of the edge and the target set is called the *head* of the edge. More formally, a hypergraph can be defined as follows:

Definition 2.4.6 (Hypergraph) *A directed hypergraph is a pair $\mathcal{H} = (V, E)$, where $V = \{v_1, v_2, \ldots, v_n\}$ is the set of vertices, and $E = \{e_1, e_2, \ldots, e_m\}$ is the set of hyperedges.*

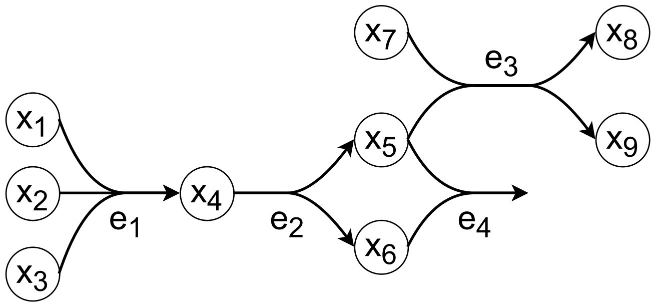
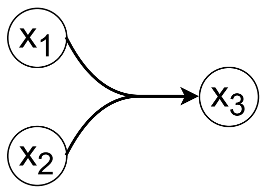


Figure 2.19: A hypergraph. Edges can have multiple sources (e_1), multiple targets (e_2) or even both (e_3). Edges can also have either an empty head or tail (e_4).

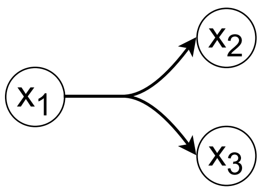
Figure 2.19 represents a hypergraph. Edges can have multiple inputs (e_1), multiple outputs (e_2) or even both (e_3). Edges can also have either an empty head or tail (e_4). A directed hypergraph is a hypergraph with directed hyperedges:

Definition 2.4.7 (Hyperedge) *A directed hyperedge is an ordered pair $e = (T_e, H_e)$, where $T_e \subseteq V$ is the tail of e and $H_e \subseteq V \setminus T_e$ is its head. T_e and H_e can possibly be empty.*

Gallo *et al.* [104] defines two different types of hyperedge. A *backward* hyperedge (B-edge) is a hyperedge $e = (T_e, H_e)$, with $|H_e| = 1$. A *forward*



(a) A backward hyperedge (B-edge).



(b) A forward hyperedge (F-edge).

Figure 2.20: Types of hyperedges in a hypergraph

hyperedge (F-edge) is a hyperedge $e = (T_e, H_e)$, with $|T_e| = 1$. Figure 2.20 represents both a B-edge (Fig. 2.20a) and an F-edge (Fig. 2.20b).

It follows that a B-hypergraph is a hypergraph whose hyperedges are B-edges. Similarly, an F-hypergraph is a hypergraph whose hyperedges are F-edges. A BF-hypergraph is a hypergraph whose hyperedges are either B-edges or F-edges.

Note that we do not need a special kind of hyperedge to represent a relation from many sources to many destinations. A hyperedge like this can be modeled using a B-edge from the sources to an intermediate node, then an F-edge from the intermediate node to the destinations. A BF-hypergraph is therefore the most general hypergraph required.

Definition 2.4.8 (Hypergraph simple path) *A path P_{st} from s to t in $\mathscr{H}$ is a sequence $P_{st} = (v_1 = s, e_1, v_2, e_2, \ldots, e_q, v_{q+1} = t)$, where $s \in T_{e_1}$, $t \in H_{e_q}$ and $v_i \in \{H_{e_{i-1}}\} \cap T_{e_i}$, $i = 2, \ldots, q$. We say that t is connected to s if there is a path from s to t.*

Gallo *et al.* [104] gives the first widely accepted definition of a directed hyperpath. Their definition is similar to the metapath definition we give in Definition 2.4.2. Later, several closely related and often equivalent definitions of directed hyperpaths were published [105–107]. The definition we give here is based on the of Ausiello *et al.* [105] .

[104]: Gallo et al. (1993), 'Directed hypergraphs and applications'

[105]: Ausiello et al. (2005), 'Partially dynamic maintenance of minimum weight hyperpaths'

Since we are interested in directed hyperpaths on directed hypergraphs, when we say hyperpath, it should be assumed that it is a directed hyperpath.

Definition 2.4.9 (Subhypergraph) *A hypergraph $\mathscr{H}' = (V', E')$ is a subhypergraph of a hypergraph $\mathscr{H} = (V, E)$ if $V' \subseteq V$, $E' \subseteq E$ and for any edge $e \in E'$ it holds that $H(e) \subseteq V'$ and $T(e) \subseteq V'$.*

Definition 2.4.10 (Hyperpath) *Let $\mathscr{H} = (V, E)$ be a directed hypergraph and let $s, t \in V(H)$. A hyperpath from s to t in $\mathscr{H}$ is a minimal subhypergraph (where minimality is with respect to deletion of vertices and edges) $\mathscr{H}' \subseteq \mathscr{H}$. Further, the hyperedges comprising $\mathscr{H}'$ can be ordered in a sequence $\langle e_1, e_2, \ldots, e_k \rangle$ such that for every edge $e_i \in \mathscr{H}$ it is the case that $T(e_i) \subseteq \{s\} \cup H(e_1) \cup H(e_2) \cup \cdots \cup H(e_{i-1})$ and $t \in H(e_k)$.*

Based on hyperpaths, hypernetworks were introduced by Volpentesta [4] who presents two types of hypernetwork, the s-hypernetwork and the (s, d)-hypernetwork. Informally, the (s, d)-hypernetwork is the subhypergraph composed of all elements in hyperpaths from a node s to a node d.

[4]: Volpentesta (2008), 'Hypernetworks in a directed hypergraph'

Definition 2.4.11 ((s, d)-hypernetwork) *Consider a hypergraph $\mathcal{H} = (V, E)$. Let $s, d \in V$ and let $\Pi_{s,d}$ be the set of hyperpaths from s to d in $\mathcal{H}$.*

The (s, d)-hypernetwork in $\mathcal{H}$ is defined as the subhypergraph $\mathcal{H}_{s,d} = (V', E')$, where $E' = \bigcup_{(\mathcal{V},\mathcal{E})\in\Pi_{(s,d)}} \mathcal{E}$ and $V' = \bigcup_{(\mathcal{V},\mathcal{E})\in\Pi_{(s,d)}} \mathcal{V}$.

The s-hypernetwork is the subhypergraph composed of all elements in hyperpaths from a node s to any node.

Definition 2.4.12 (s-hypernetwork) *The s-hypernetwork in $\mathcal{H} = (V, E)$ is $\mathcal{H}_s = \bigcup_{x\in V} \mathcal{H}_{s,x}$.*

2.4.3 Metagraphs, Hypergraphs and policies

[103]: Basu et al. (2007), *Metagraphs and their applications*

The foundational text on metagraphs is the book *Metagraphs and their Applications* by Basu and Blanning [103] .

[28]: Ranathunga et al. (2020), 'Verifiable Policy-Defined Networking using Metagraphs'

To the best of our knowledge, metagraphs belong to the rare appropriate structures able to naturally model access control policies. This theoretical construct guides the reasoning about interactions between policies [28] , and we argue they are the best constructs to deal with. Policies are generally modeled with sets (of, e.g., users, resources) and relations between them, thus allowing for graph modeling. Basic directed graphs, however, do not handle well sets of elements while more advanced structures like simple hypergraphs do not have any sense of direction of their edges. Other solutions like SAT solvers might have formal foundations, but they lack a visual representation of the policy. Basu *et al.* [103] discuss graphs as well as alternative structures such as petri nets and higraphs, and consider their shortcomings when compared to metagraphs. The closest structure to metagraphs is the directed hypergraph: the main difference between the two lies in the purpose of the analysis done with either [103].

Additionally, four main paradigms exist to model processes and their workflows, and metagraphs can integrate three of them [103]. Informational modeling focuses on the informational entities involved in the process, e.g. the inputs and outputs of tasks in the process. Functional modeling is more concerned with the relationships between the different tasks of the process, e.g. which outputs of a task serve as an input to another task. Organizational modeling focuses on the agents/resources related to the tasks, e.g. which agents are assigned to which tasks. Finally, the focal point of transactional modeling is the issues of timing and sequencing of the tasks, e.g if the tasks are realized in a sequence or in parallel. Metagraphs can integrate the informational, functional and organizational modeling paradigms in a single model. Furthermore, some issues of timing and scheduling pertaining to the transactional model can also be addressed using metagraphs.

Even though we argue metagraphs are one of the most suited tool for representing and reasoning about policies, they are still underused with only few existing works in the literature. Metagraphs have been used to represent and verify business processes [108], to verify policy consistency in MUD profiles for IoT devices [109], to detect redundancies and conflicts in network policies for distributed firewalls [28] , and even to reconcile local E-health policies with the GDPR legislation to identify violations and omissions [110]. In Chapter 4 and in opposition to those works, we verify a policy implementation matches its specification, and evaluate our method.

[28]: Ranathunga et al. (2020), 'Verifiable Policy-Defined Networking using Metagraphs'

Ranathunga *et al.* [111] defined a toolkit in python to manipulate metagraphs. Hamza *et al.* [109, 112] use metagraphs to model policies in IoT devices to generate and validate Manufacture Usage Descriptions (MUD) profiles—MUD profiles can be used to define the access control model and network functionality these devices need to properly function. They also check compliance of those MUD profiles with different levels of security policies, to determine where those devices are safe to be deployed.

[111]: Ranathunga et al. (2017), 'MGtoolkit: A python package for implementing metagraphs'

Directed hypergraphs have also been widely studied [104, 106, 113–116]. Basu and Blanning [103] mention that metagraphs are related to directed hypergraphs, however this relationship does not appear to be explored at any significant depth.

2.5 Conclusion

In this section, we first presented workflows. We gave workflows a working definition, and showed how they were composed by explaining the different elements one can use in the YAWL language. We also presented components usually found in microservices. Namely, we detailed containers, and showed how they were used in an orchestrator. We explained how a service mesh works, and illustrated the use of policy engines as sidecars. We introduced the main archetypes in use when it comes to authorization. Finally, we presented metagraphs and hypergraphs by defining them and explaining how they worked. For each of those themes, we presented related works close to our contributions.

A Secure Infrastructure to Prevent Data Exposures 3

In the last chapter, we reviewed concepts pertaining to our contributions.

This chapter is dedicated to our secure infrastructure, and will mainly lean on the concepts of workflows and microservices. We first describe the problem at hand, that is how to create an infrastructure that is able to support our security requirements for workflows. We detail the properties we require, as well as our assumptions. Those assumptions constitute our threat model, our trust model as well as our attacker model. Next, we detail our designed overall infrastructure which meets our requirements. We describe our Proof of Concept as well as a method to verify policies are correctly enforced. We finish by detailing the overhead cost of the security benefits of our infrastructure.

3.1 Problem statement

In the context of a workflow, the system we want to achieve should enable the secure exchange of data and its security at rest while avoiding any leak.

As a reminder, we define a workflow as a sequence of tasks to be performed by a set of independent actors. The owner of the data (i.e., the instigator of the workflow) interacts with contractors to realize such a sequence. The workflow is defined by the owner, which defines how and by whom the data he possesses should be processed and specifies the different steps needed to achieve his objective. Those tasks are realized by contractors, which perform the task they have been assigned to, on the data they have been given. Both the owner and the contractor have agents processing the data, where agents can represent an employee or an automatic service. Contractors possess some business intelligence, which we define as the tools and methods used to fulfill their task(s).

The owner has ownership over the data being processed: he does not want his data to be leaked in any way. On the other hand, the contractors do not want the other actors involved in the workflow, including the owner, to learn about their business intelligence.

To illustrate, we use the same recurring example pertaining to the movie industry (Chapter 2, Figure 2.1). Let us consider the case of a workflow, where an owner in the post-production stage of making a movie wants to employ other companies to edit the video and audio components of the movie [17]. More specifically, let us imagine that for example the owner wants to add special effects, tune colors, set up High Dynamic Range (HDR) and master the audio. In particular, he wants the application of the special effects first, and then the color tuning, and, finally, HDR in parallel with the sound mastering.

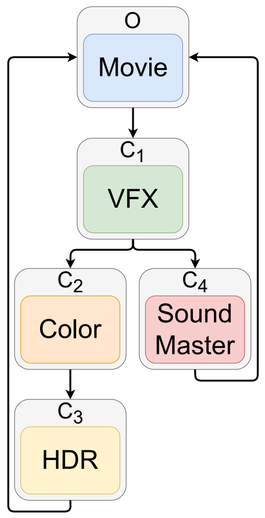


Figure 3.1: Movie workflow example. Arrows model the specified workflow, and thus represent the communication flow of our example. The owner (O) sends its data to the first contractor (C_1), for special effects processing. C_1 then sends the modified data along the workflow, for color (C_2), HDR (C_3) and sound (C_4) processing. The resulting data is then sent back to the owner.

The intent of the owner can be modeled under the form of a workflow, that is a directed acyclic graph as depicted on Figure 3.1. The owner (O) first sends its data to the company responsible for special effects (C_1). C_1 applies special effects to the movie sequences the owner sent him, and then sends the result to the company responsible for coloring (C_2) as well as another for sound mastering (C_4). C_2 then ships its result to the agent in charge of High Dynamic Range (HDR) (C_3). Finally, both C_3 and C_4 sends their output back to the owner, which puts the data together to obtain the final product.

The purpose of our solution is to allow this workflow to take place, with the movie transiting from the owner (O) through the contractors (C_1, C_2, C_3, C_4) back to the owner, while guaranteeing data security at rest and in transport. With our proposal, we prevent unwanted communications at the network and application level.

3.2 Threat and security model

3.2.1 Properties and threat model

With our solution, we want to guarantee the following properties.

- **Data security at rest**: Data within the workflow is stored encrypted (confidentiality) to prevent malicious entities from reading the data. Access to the data is restricted by isolation. This data cannot be leaked outside of the workflow, before, during or after the workflow is terminated.
- **Data security in transport**: Data is exchanged encrypted (confidentiality) to prevent eavesdroppers from reading the data. The data is also transmitted accurately and complete (integrity), with a verified origin and destination (authentication). This data cannot be leaked outside of the workflow, before, during or after the workflow is terminated.

We consider a threat model from the point of view of each actor of the workflow.

- **Owner**: The owner does not want the data it sends to the contractors involved in the workflow to be leaked. The threat here is that an agent leaks the data of the owner to an unauthorized party, or that the data is accessed by an unauthorized adversary.
- **Contractor**: The contractors do not want the other actors involved in the workflow to learn about their business intelligence. The threat in this case is that actors learn about the processes used by a contractor. For example, actors might be able to guess the set of actions applied on the data or the algorithms used to transform the data.

3.2.2 Trust model: actors and environment

From the point of view of the data owner, trusting the contractors is one thing, trusting its agents another. In other words, if the owner trusts the organization of the contractor to not intently bypass our system, controlling the actions of the contractor's agents is then possible for both the owner and the contractors. If one does not trust the contractors to deploy the infrastructure they are required to deploy, there is no easy way to verify that the data is actually sent to the secure environment we designed (Sec. 3.3), therefore removing any guarantee we might have concerning data leaks. Removing this trust between actors has its own drawbacks in a real world deployment.

In contrast, looking at a finer granularity, actors do not need to trust their agents and the ones of the other actors. Even though agents are deterred from engaging in malicious activities, due to the nature of their relationship with their companies (internal rules, non-disclosure agreements, ...), they can still put data and/or business intelligence at risk through accidental exposure or malicious behavior. Actors are thus assumed to be malicious. Our solution controls those agents to prevent owner data and business intelligence leaks. This is consistent with our need to trust the contractors, since business to business contracts have the same deterrents, but with much higher stakes at play.

From the point of view of a contractor, we have the same trust issues as the owner. Other actors, including the owner, might try to reverse engineer the business intelligence of the contractor. This reverse engineering process requires a lot more effort than simply having access to the data of the owner, and might prove to be very hard or even impossible to do in some cases[1] . In the same way the owner needs to trust that contractors do not intently bypass our system, the contractors need to trust that actors sending them data do not tamper with it. Like the owner, contractors do not trust the agents.

1: This can happen in very specific cases, such as when a contractor receives its input(s) and gives its output(s) to the same actor. As data is encrypted in transport, only the two ends of a communication see the data. A solution would be to insert the owner between contractors such as to limit their learning of the workflow and trust the owner not to reverse engineer the actions of its contractors.

Finally, both the owner and the contractors need to trust the owners of the environments involved in the workflow. While the environment an actor is using can be owned by this actor, meaning the added trust requirement is the same as trusting the actor, some actors can use a third-party environment to fulfill their task(s) (e.g. a cloud provider). Since this third-party provides (part of) the environment the workflow will be deployed on and has admin rights to the machines supporting the workloads, it can try to gain access to the data of the owners or the business intelligence of the contractors. We would need to enhance our solution with Trusted Execution Environments (TEEs) in order to fully remove the need for trust in those third-parties. With the proposed infrastructure, one needs to trust those potential third-parties. As such, actors and environment providers are considered honest but curious.

To summarize, from the point of view of the owner or a contractor, we trust everything but the agents. Actors are assumed to be honest but curious, while agents are assumed to be malicious.

3.2.3 Attacker model: external attackers and malicious Agents

Taking into account the assets to protect and the trust model, we consider three types of attackers in our model. As our solution is in the form of a deployed infrastructure, an attacker can be internal or external to the modeled workflow. An external attacker can then either be co-located with the deployed workflow (e.g. the attacker is located in a cloud partially or fully hosting the workflow) or external, attacking from a remote location.

- **External attacker**: External to the workflow and the location of the infrastructure deployment. This highly motivated and often technically skilled attacker tries to gain access to the data or the business intelligence of the contractors from the outside.
- **Co-located attacker**: External to the workflow, but co-located at the deployment (e.g. an attacker located in one of the clouds that are part of the workflow deployment). This highly motivated and often technically skilled attacker tries to gain access to the data or business intelligence of the contractors, but from a co-located position that opens more exploit possibilities.
- **Malicious agent**: Internal to the workflow, this attacker tries to leak the data outside of the workflow.

Despite the fact our model already covers most cases, it does not deal with the full range of possible attacks. Fully protecting against some attacks (e.g. from a contractor or a third-party cloud provider) would make the system less convenient and usable for contractors or the owner. Protection from leaks resulting from physical attacks such as when an employee takes a picture of his screen are not considered.

3.3 Overall description of the infrastructure

We now present the infrastructure we propose for protecting a workflow execution from the threats expressed in Sec. 3.2. As we need a way to prevent data leaks, we need to control the communications an agent can engage in. To achieve this, we need to control the environments the agents will be using, to make sure that all the actions of an agent follow a policy enforced by the owner. We opted to do this using the microservice architecture, for the benefits granted by the components of the infrastructure listed below. Besides, they are already commonly deployed to provide many services like automatic scaling and isolation.

In this infrastructure, agents of our workflow are mapped to containers, which are then used in conjunction with an orchestrator, a service mesh and policy engines to enforce the policy of the owner.

Figure 3.2 shows the workflow we defined in Figure 3.1, with each actor having its own deployment space represented by the cloud surrounding the boxes which represent the agents of those actors (e.g. the C_{1_1} box represents an agent of contractor C_1). The access policies of a service are pushed in the policy sidecar associated with the service.

Figure 3.2 also illustrates how we use the elements of the microservice architecture. Each agent is a pod, containing the service (i.e., the environment the agent will be using), a proxy and a policy sidecar. The proxy sidecar will intercept all traffic coming from and going to its respective service. The proxy will then check thanks to the policy sidecar if the request is authorized or not. If the request is authorized, it is forwarded accordingly, and the request is rejected otherwise.

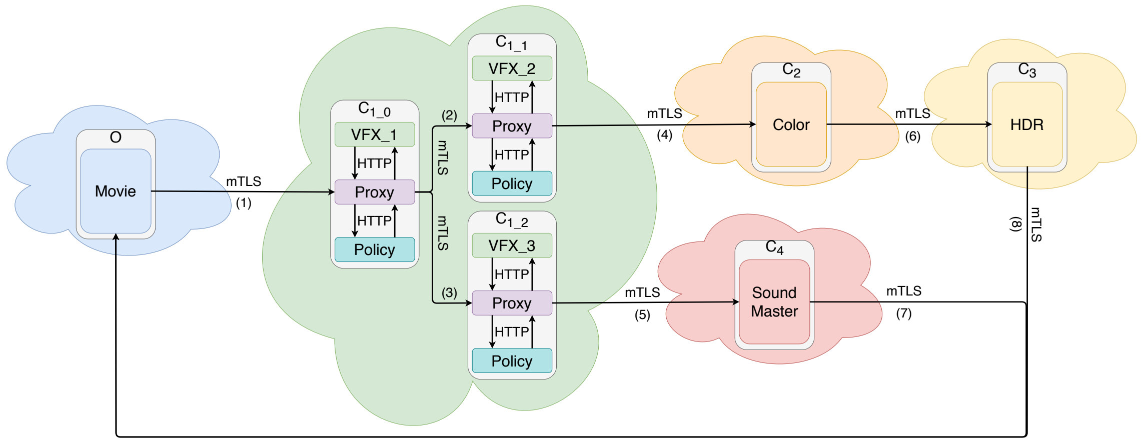


Figure 3.2: Secure infrastructure. Each box represents an agent. It is a pod with the appropriate containers. The container of the color of the actor represents the service. Purple containers represent the proxies of the service mesh, and blue containers represent the policy sidecars. The arrows stipulate whether the communications are secure (`mTLS`) or not (HTTP).

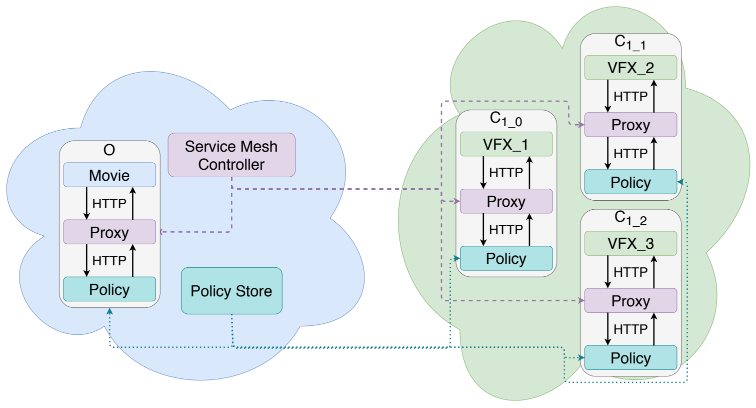


Figure 3.3: Representative subset of the secure infrastructure control plane (contractors C_2 through C_4 are not represented). Proxies are configured by the service mesh controller, providing them with identities and key pairs, as well as routing information for them to initiate secure communications with other proxies in the mesh. Policy changes are enabled with periodical pull on the policy sidecars (whose input comes from a policy store).

Proxies are configured by the service mesh controller (Figure 3.3), providing them with identities and key pairs, as well as routing information for them to initiate secure communications with other proxies in the mesh. Policy is pulled periodically by the policy sidecars from a policy store, which allows for policy changes. Since the service mesh controller and the policy store are under the control of the owner, he is in control of the system. Thus, the owner specifies the policy to be applied to enforce the desired workflow, preventing data from leaking outside.

The data processed by the pods is stored on mounted Persistent Volumes (PVs), which are encrypted with a key located in a key-value store of the orchestrator, providing us with **data security at rest**. We generate a key to encrypt each PV required by needs of the workflow. Since the keys are all stored in the same key-value store, this does not really mitigate risks against a technically skilled attacker gaining access to the key-value store, but it can help to protect some of the data in case the attacker only gains access to a subset of the keys through other means. Since the keys are stored in the master components of the orchestrator, they are under the control of the owner. To enforce the workflow and make sure the agents cannot bypass it via the PVs, each agent must have its own personal PV.

Pods can also communicate according to the specified workflow and policy via `mTLS`, providing us with **data security in transport** as indicated by the communications between the pods in Figure 3.2. Communications inside a pod are not encrypted, but the isolation layers protect the data against eavesdroppers.

Once a service has completed all the tasks he was assigned to do, the associated pod is destroyed to make sure data cannot be leaked from this service past this point in time. To provide data security in transport, services in the service mesh are provided with an identity in the form of a certificate, which is associated with a key pair. To make sure those are safe to use, and that no attacker gained access to the keys or tampered with them before they reach the appropriate service, we need to verify the key distribution process is secure. In the case of the service mesh, this is done automatically for us. Appendix A.2 summarizes this procedure, and how it is secured. Thanks to this infrastructure, communications can be constrained to follow a policy, giving us a streamlined way to prevent data leaks. We show how a simple policy to prevent data leaks can be defined.

3.4 Proof of concept

We realized a proof of concept, by implementing the infrastructure described in Sec. 3.3. We reproduce the workflow of Figure 3.2, with services of the workflow receiving and sending arbitrary data to represent the data of the owner. We use Docker [23] for our containers, Kubernetes [24] for our orchestration layer, Istio [25] as our service mesh, using Envoy [26] for the proxy sidecars and Open Policy Agent [27] (OPA) for the policy sidecars. We also use Kubernetes to provide the services with encrypted volumes. This infrastructure was deployed on Google Cloud Platform (GCP), using one cluster for each actor of the workflow, for a total of five clusters. Each cluster runs one **n1-standard-2** node (2 vCPUs, 7.5 GB of memory), on version **1.14.10-gke.36**, except the cluster of the owner which runs two of them, since running the control plane requires additional resources. The clusters for the owner, color and VFX are located in us-central1-f whereas the clusters for HDR and sound are located in us-west2-b.

The workflow we want to enforce is shown in Table 3.1, where each row represents the **source** of a request, and each column a **destination**. The full policy is not represented by this table, additional attributes further constrain those permissions (see Listing A.1 for the full policy). The agents can also send GET requests, but they are all denied by the policy.

The complete data, code as well as guidance to realize this Proof of Concept are publicly available[2] .

2: See `https://github.com/loicmiller/secure-workflow`.

We also developed a test framework to check that:

- ▶ Traffic is either encrypted or protected inside a pod by the isolation provided by the pods;
- ▶ The policy, allowing or denying communications between services, is correctly enforced.

Table 3.1: Proof of Concept policy.

	Destination						
Source	owner	VFX_1	VFX_2	VFX_3	Color	Sound	HDR
owner	-	POST					
VFX_1		-	POST	POST			
VFX_2			-		POST		
VFX_3				-		POST	
Color					-		POST
Sound	POST					-	
HDR	POST						-

To do so, we capture traffic on every network interface in the service mesh and perform each possible communication. In the general case, considering we have N services and M types of request, we obtain the number of possible communications with the formula: $N(N-1)M$. Since we capture on each interface, and services have a loopback as well as an external interface, we obtain the total number of required captures: $N(N-1)M(2N) = 2(N^3 - N^2)M$. The number of required captures thus grows cubicly with the number of services and linearly with the number of requests.

Considering our previous example in Sec. 3.4, we have seven services, two possible requests (GET and POST), which gives us a total of 1176 captures. Captures are obtained from a `tcpdump` container added to the service pods as a sidecar. Since containers in the same pod share the same network namespace, capturing traffic from the `tcpdump` container on either the loopback or the external interface allows us to see traffic from the other containers in the pod. Figure 3.4 shows the path a communication takes inside the service mesh, as well as whether traffic is encrypted or not. The request is initiated by service A, and intercepted by its associated proxy via the loopback interface. The proxy will then check thanks to the policy sidecar if the request is authorized or not. If the request is authorized, it is forwarded accordingly. The request is rejected otherwise. In the case where the request is authorized, it is forwarded to the proxy of service B by using `mTLS`. There, the proxy forwards the request to service B, which replies by going through the same steps as earlier. Traffic going to/coming from the loopback should be unencrypted, whereas traffic going to/coming from the external interface should be encrypted. Our captures show that this is indeed the case. Traffic does not need to be encrypted on the loopbacks, as all the elements (i.e, the service and its sidecars) that have access to this loopback are in the same trust zone. The layers of isolation provided by the pods protect the loopback traffic from being seen by unauthorized entities.

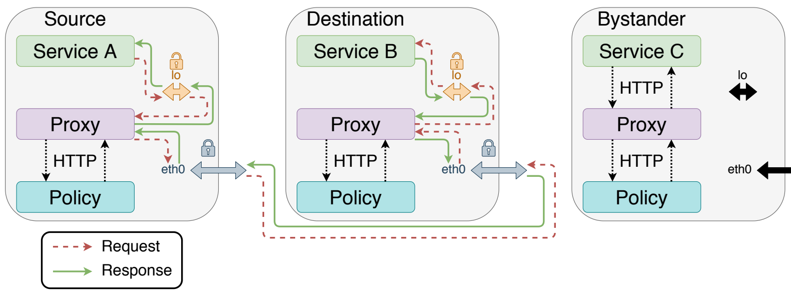


Figure 3.4: Detailed view of pods and the communication flow. Traffic is unencrypted on the loopbacks, but encrypted on the external interfaces.

Obviously, pods in the service mesh have one of three roles during a communication. Either they are the source of the communication, the destination of the communication, or simply a bystander that is not involved in the communication. This is important, because the checks we need to perform depend on where traffic was captured:

- **Source/Destination loopback**: We need to verify that a communication between the source and the destination is occurring (i.e., correct IP addresses and ports). We need to verify that the request in the capture corresponds to the request we are testing for (GET or POST). The response needs to be in accordance with the policy: in this case, '403 Forbidden' if the policy was 'deny' and '200 OK' (GET) or '201 OK' (POST) if the policy was 'allow'.
- **Source/Destination external interface**: We need to verify that a communication between the source and the destination is occurring (correct IP addresses and ports). We need to verify that the traffic is encrypted by `mTLS`, and not passed in clear text.
- **Bystander loopback and external interface**: We need to verify that no communication between the source and the destination is occurring, whether encrypted or unencrypted.

We built a tool that automatically extracts the authorization policies from the OPA policy configuration, generates and then tests an access control matrix. For each possible communication in the service mesh, our tool loads all the captures relevant to this communication, identifies them to see what we should verify in each capture, and then proceeds to check if captures are in accordance with the criteria above. It is then easy to evaluate whether the system is compliant with the overall policy. The complete code for the test framework is publicly available[3] .

3: See `https://github.com/loicmiller/secure-workflow`.

3.5 The overhead of security

In this section, we analyze the performance overhead added by the policy sidecar enforcing security. We measure the pod startup time, and the request duration (between each couple of connected pods).

3.5.1 Startup time

We first evaluate the impact of having an additional container for OPA on the startup time of pods. An independent-samples t-test was conducted to compare startup times in a deployment of our PoC with or without OPA. Our aim is to determine whether these is a significant statistical difference in startup time when OPA is instantiated. For that purpose, we test two hypotheses. (H_0) There is no statistically significant difference in the startup times of the two deployments, and (H_1), a statistically significant difference exists between the startup times of the two deployments.

We gathered 130 observations per pod per deployment ($N = 1820$ total). We measure the startup time of pods by scraping the transition timestamps between `PodScheduled` and `Ready` states of the pods.

Figure 3.5 allows to compare the distribution of startup times in the deployments with and without OPA to measure the cost on the initial deployment.

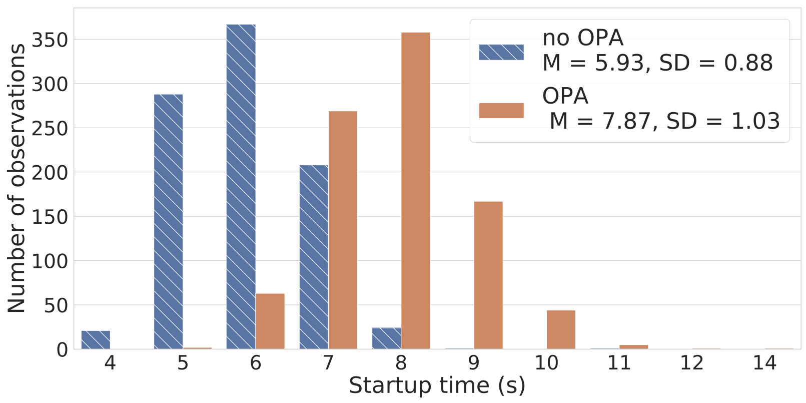


Figure 3.5: Distribution of startup time in the deployment without OPA (stripes) compared to the deployment with OPA (full)

The 910 observations in the group with the OPA sidecar ($M = 7.87, SD = 1.03$), compared to the 910 observations in the group without the OPA sidecar ($M = 5.93, SD = 0.88$), exhibit significantly higher startup times, $t(1818) = 43.19, p < 0.001$. The *effect size* for this analysis, $d = 1.985$, was found to exceed Cohen's convention for a large effect ($d = 0.80$). Running a *post hoc power analysis* also reveals a high statistical power, $1 - \beta > 0.999$.

These results show that pods with OPA have a substantial increase in startup time of almost two seconds on average. This corresponds to an increase of 32.72% of the startup time. We ran the same analysis on a per pod basis and found the results to be consistent with those results.

3.5.2 Request time

This second evaluation is about measuring the influence of the policy size on communications, to test whether the policy is scalable for more complex workflows. We analyze separately requests happening within the same region (intra-region e.g. us-central1-f to us-central1-f) from inter-region requests (e.g. us-central1-f to us-west2-b) as communication times are significantly different in the two cases.

For each request duration between pairs of pods, we take into account whether the authorized communication happened inside a region or between regions (inter-region), since the duration of requests are different for those types of communication, and thus the impact of policy size is not the same.

A *one-way between subjects ANOVA* was conducted for each type of communication (intra-region and inter-region) to compare the effect of policy size on request duration in five levels of policy size: `no opa`, `all allow`, `minimal`, `+100` and `+1000`.

The `no opa` policy deployment corresponds to having no OPA container at all. The `all allow` policy deployment corresponds to having no rules and allowing all communications by default. The `minimal` policy deployment corresponds to having the default minimal number of rules to enforce the workflow of the PoC. The `+100` and `+1000` correspond to the minimal policy being inflated respectively with 100 additional rules (+147%) and 1000 additional rules (+1470%), with additional rules being obligatorily evaluated by OPA.

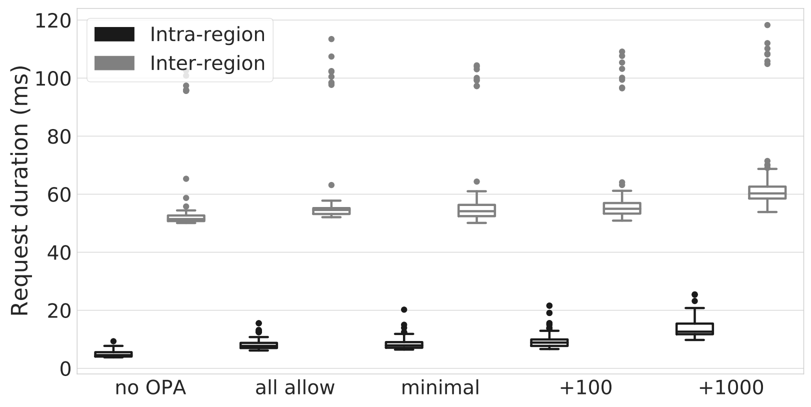


Figure 3.6: Spread of request duration for intra and inter-region communications by policy size.

For each ANOVA, we gathered 40 observations per authorized communication per level of policy ($N = 1600$ in total). We measure request duration by recording the elapsed time of requests between pairs of pods via cURL.

Figure 3.6 shows the distribution of request duration for each policy size. For intra-region communications, there is a significant difference in request duration among the five scenarios of policy deployments, $F(4, 795) = 364.05, p < 0.001, \eta^2{}_p = 0.65$. Post hoc comparisons using Tukey's HSD test indicated significant differences between the mean scores of all policy deployments, e.g. `no opa` ($M = 0.0049s, SD = 0.0010$), `+1000` ($M = 0.0136s, SD = 0.0028$), except the `all allow` and `minimal` deployments.

For inter-region communications there also exists a significant difference (in request duration) among the five scenarios of policy deployments, albeit with a lesser effect: $F(4, 795) = 15.23, p < 0.001, \eta^2{}_p = 0.07$. Post hoc comparisons using Tukey's HSD test indicated that the mean scores for policies close in size were not significantly different (e.g. `no opa` and `all allow`), whereas policies distant from each other in terms of size were found significantly different (e.g. `+100` and `+1000`)[4] .

4: See `https://github.com/loicmiller/secure-workflow` for full data, code and statistical analysis in the form of jupyter notebooks.

As expected, the impact of policy size is weaker in the inter-region communications as those communications are naturally longer and more subject to variations than intra-region communications. Policy size accounts for 65% of the variance in intra-region communications whereas it accounts only for 7% of the variance in inter-region communications.

Taken together, these results suggest that policy size does have a considerable effect on request duration. While our results suggest that a higher policy size increases request duration, it should be noted that the size must be quite high in order to observe an effect, especially regarding inter-region communications. With such communications, incremental increases in policy size do not appear to have a significant effect on request duration.

3.6 Discussion

In this chapter, we showed how to use the microservice architecture to secure workflows. Our infrastructure enables the secure exchange of data and its safety at rest while avoiding any leak. We defined our threat and security models, stating we want to guarantee data security at rest and in transport. More specifically, in accordance with the principles of zero-trust, we achieve a secure system that enables the exchange of data between non-trusted agents while guaranteeing this data is secure at rest, in transport and cannot be leaked by any agent in both cases, in the context of a workflow. In our trust model, actors are assumed honest but curious whereas agents are assumed to be malicious.

We then described our infrastructure, and detailed our proof of concept. With our model and proof of concept, we provide **data security at rest** by using encrypted volumes as well as multiple layers of isolation in the form of pods and containers. We also provide **data security in transport** by using mTLS for the communications that are exposed, which provides us with encryption, authentication and integrity of the data being exchanged. The workflow is defined by the owner and enforced using policy sidecars, which controls the agents participating in the workflow.

Looking back at our attacker model, one can see how the elements of the infrastructure prevent adversaries from attaining their goal. The malicious agent cannot leak data outside of the workflow, since he is isolated in a pod, and all requests emanating from the agent are intercepted and submitted to the implemented policy, which prevents him from leaking data. Communications the agent makes are encrypted, and can only be performed with other authenticated agents, preventing eavesdropping and impersonation. Since services are isolated and can only be reached by going through the proxy (and its associated policy sidecar), an external attacker cannot perform any data breach. The co-located attacker faces the same issue, and system exploits may be prevented via the isolation provided by pods and containers. Obviously, protection is only provided if the trust model is respected, and the policy is defined and implemented correctly. The owner has all the tools in hand to do so, since the service mesh controller and the policy store are under its control.

Using our solution comes with other practical advantages. Our infrastructure is easy to use as each actor can opt for the environments of its choice in order to deploy the workflow. The standardized environment grants reduced complexity for security management, compared to monolithic solutions. Changes to the workflow and the policy can be applied at the centralized policy store, using a high-level policy language.

The business intelligence of the actors is separated from the infrastructure; since our architecture model is modular, one can choose which solution to use for each part of the infrastructure. A service mesh can be swapped for another, same thing for the orchestrator and the container technology, meaning there is no vendor lock-in, the solution is flexible.

The service mesh also provides fine-grained telemetry. This observability feature (e.g. metrics, logs, tracing and service mesh visualizations) allows to monitor actions of the agents in the pods. One can use numerous addons to Istio to improve the monitoring of the system (e.g. Kiali, Jaeger, Zipkin).

To the best of our knowledge, this is the first time someone showed how to use the microservice architecture to secure workflows. Though we manage to secure data at rest and in transport using elements of the microservice architecture, some threats are still potent. First, using our infrastructure does not prevent any attacks on the services that are deployed in it. For example, someone using an unsecure outdated version of *apache* within our infrastructure will be vulnerable to exploits possible on this version of the service. Since the service inside our infrastructure can be literally anything, there is little we can do to prevent those attacks. Second, our infrastructure does not prevent any side-channel attack. Third, we assume actors to be honest but curious, and we trust the platform(s) the workflow is deployed on. This assumption does not always hold, as the underlying platform or even actors can be compromised or malicious.

Finally, our infrastructure assumes its policy specification to be correct, and its implementation to be exact. However, this is not always the case, so our infrastructure is open to attacks using the fact the policy is not always correct. Indeed, a policy specification can contain errors or redundancies, and it does not always match its implementation as errors can be made in the translation process, whether by a faulty tool or human error. To address this limitation, we propose in Chapter 4 a way to verify that the policies deployed in our infrastructure actually match their initial specification, and in Chapter 5 a way to verify the initial specification does not contain any redundancies.

3.7 Conclusion

In this chapter, we set out to find how to use the microservice architecture to secure workflows. We defined our threat and security models, stating we want to guarantee data security at rest and in transport. We then described our infrastructure, and detailed our proof of concept. Using the Google Cloud Platform, we deploy a sample workflow and verify the policy implementation is actually enforced via basic captures at crucial locations in the infrastructure. We finally measured the overhead of security by comparing startup times of containers with and without Open Policy Agent, as well as comparing request times according to the size of the deployed policy. The added security provided by the workflow enforcement costs pods two seconds of startup time on average, and either 7% or 65% of the variance in request duration.

This work has led to two publications [19, 20] and the development of a publicly available Proof of Concept.

Overall, we recommend following the appropriate guidelines when deploying a workflow using microservices, and advise to examine the considerations laid out in this in this chapter when it comes to protecting the data. Possible future works include studying how changes in the workflow impact the security of our infrastructure, or trying to remove some of the trust requirements. As we said, the underlying platform or even actors can be compromised or malicious, and lowering our trust requirements would make our overall infrastructure more secure. A possible lead to lowering those requirements would be to use Trusted Execution Environments (TEE), which are a secure area inside a processor. Code executed inside a TEE is encrypted, and cannot be accessed even by the underlying platform, providing confidentiality benefits. Using TEEs, tasks running inside our workflow would be protected even from the cloud provider those tasks run on. Further works are needed to study both the security and limitations of TEEs. For one, the paging system of TEEs imposes a size limit, which might be prohibitive for workflows exchanging large amounts of data.

Verifying Policies Using Metagraphs 4

In the previous chapter, we reviewed how to secure data at rest and in transport using our secure infrastructure. Here, we explain how to perform policy verification using metagraphs.

To perform our policy verification, we propose to model policies with a generic but rich enough structure, metagraphs. We use their formal foundations to verify whether the implementation of a policy, its actual deployment, matches its initial specification.

We rely on this structure since, by design, it provides means to correct conflicts and avoid redundancies [28] , as well as a more efficient verification process than with other structures (e.g. usual graphs). We will expand on how metagraphs can be used to detect redundancies in Chapter 5. More general than simple graphs, metagraphs are especially indicated to model workflow policies [103] .

[28]: Ranathunga et al. (2020), 'Verifiable Policy-Defined Networking using Metagraphs'

[103]: Basu et al. (2007), *Metagraphs and their applications*

We specifically demonstrate their benefits in the case of workflows. AS discussed in Chapter 2, processes and their workflows are commonly modeled using one of four modeling paradigms, each of them focusing on one specific dimension of the workflow, while metagraphs can integrate three of those paradigms within a single model. Metagraphs can also model the scheduling of time-critical workflows, which are handled by the propositions of the conditional metagraph.

First, we will describe the advantages as well as how to use a metagraph to model workflows and processes. Then, we will explain how we use metagraphs to verify policies, and we will go through the transformations we need prior to the verification process. Finally, we will show a performance analysis of our method and conclude.

4.1 Metagraphs as a model for workflows

Metagraphs are an efficient specification tool to model processes and their workflows. They integrate all the informational, functional and organizational modeling paradigms within a single model. Considering Yet Another Workflow Language [47] as the most representative tool to specify, analyze and execute workflows today, we will show how to transform any of its modeling components (e.g. composite tasks, conditions) into a metagraph representation. To help explaining this translation, let us consider the film production process we presented in Figure 2.7. We remind the reader this case study represents the chain of information processing realized along the shooting of the movie [49]. Without a loss of generality and for simplicity purposes, we only consider the start of the process, represented in Figure 4.1. Figure 4.2 shows the same part of the process, translated to a metagraph.

[47]: Van Der Aalst et al. (2005), 'YAWL: yet another workflow language'

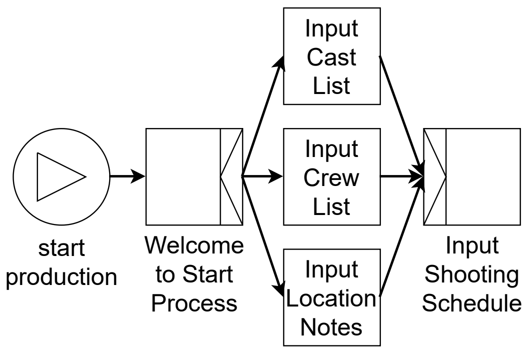


Figure 4.1: Film production process represented in YAWL. Only the start of the YAWL process is represented for brevity (Figure 2.7).

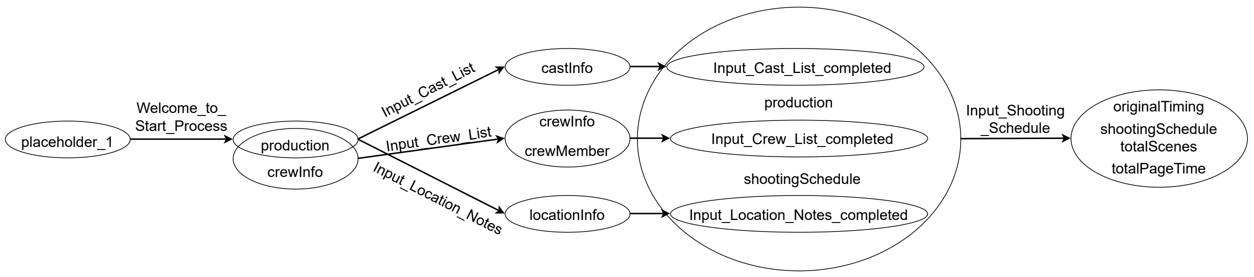


Figure 4.2: Film production process represented with a metagraph. The task `Input Cast List` does not share variables with its next task, `Input Shooting Schedule`, thus we add a proposition indicating the completion of the task.

Tasks and nets

In a metagraph, we can model tasks as edges of the metagraph, where the inputs (outputs) of a task correspond to the invertex (outvertex) of the edge. Using metagraphs, we can model atomic tasks as edges, and composite tasks can be edges representing the composite tasks, with each composite task being its own sub-metagraph, mirroring the nets in YAWL.

Processes and workflows

A process may contain information elements which are not yet evaluated. A workflow is an instantiation of a process for a set of particular values. Thus, a process can result in multiple workflows. Taking into account this definition, a process can be modeled by a conditional metagraph, with all of its propositions still not evaluated. A metagraph with no propositions, i.e. a simple metagraph, can represent a workflow, i.e. one particular instantiation of the process [103].

Roadblocks in the modeling

In YAWL, a task can have any input and any output, or can even have no input or output at all. At first glance, this seems to limit our model in two ways. First, since we model tasks as edges, tasks with no input (output) have no invertex (outvertex). To overcome this limitation, we add placeholder variables to empty vertices, which have no effect, but extend our modeling abilities to include such tasks. Conditions are also modeled using placeholder variables since they do not have any input or output.

Second, in YAWL, two tasks taking place sequentially in the process are not necessarily linked by their variables. In other words, the outputs of the first task do not necessarily correspond exactly to the inputs of the second task. This may look troublesome, since our metagraph representation relies on those links to retain the information of the workflow (i.e. which task should be performed next?), and makes use of paths and metapaths in its analysis. To enforce the sequence of tasks, we need to add propositions to the metagraph in certain cases.

There are two possibilities when considering two tasks chained sequentially, regarding the outputs of the first, and the inputs of the second. The first possibility is that the outputs of the first task correspond at least partially to the inputs of the second task, that is at least one variable is common to the two sets. In this case, no propositions need to be added since we have at least one shared variable, modeling the link between those two tasks. The second possibility is that there is no correspondence between the outputs of the first task and the inputs of the second task – the tasks are disjoint. For example, this occurs when the outputs of the first task are not necessary to perform the second task, but we still want the tasks to be executed in this specific order, like the `Input Shooting Schedule` task of Figure 4.1. In this case, we need to create a new edge from the outputs of the first task to a proposition signifying the first task has been completed. Once this is done, we can add this proposition to the input of the second task, to preserve the link between the tasks. This is shown in Figure 4.2, where the task `Input Cast List` does not share variables with its next task, `Input Shooting Schedule`. If the proposition is true, the task has been completed and the second task can start – the task becomes unavailable otherwise. Thus, the proposition clarifies the fact that `Input Cast List` needs to be completed in order to proceed to the `Input Shooting Schedule` task.

Operators

The `AND`-split, `OR`-split and `XOR`-split elements are handled naturally in a metagraph. Since each task is represented by an edge, a split simply corresponds to multiple edges sharing the same invertex. However, modeling the `AND`-join, `OR`-join and `XOR`-join elements is a bit more complex.

With join operations, we cannot say it simply corresponds to multiple edges which share the same outvertex, since, depending on the operator, we might want all or only some of the tasks preceding the join to have been executed. The `OR`-join requires at least one of the preceding tasks is executed, so no additional measures need to be taken. An `AND`-join on the other hand requires all tasks preceding the join to be executed, whereas a `XOR`-join requires only one of them is. To model this intent, we add completion propositions to the metagraph, in the same way we did for disjoint tasks. Those propositions are illustrated in Figure 4.2, where the tasks `Input Cast List`, `Input Crew List` and `Input Location Notes` are AND-joined in the next task, `Input Shooting Schedule`. It follows that the `AND`-join needs all propositions to be true, whereas the `XOR`-join needs exactly one of them to be true. To check for the fact all other propositions are false in the case of the `XOR`-join, we create non-completion propositions in addition to completion propositions. A non-completion proposition is true if the task has not been executed, and false otherwise.

Using this metagraph representation brings us many advantages. First of all, it is easier to identify and analyze workflows associated with a process, via the evaluation of the propositions in the conditional metagraph representing the process. This representation can also help us analyze the independence of decomposed subprocesses, as well as the redundancy and full connectivity of composite processes via the union of metagraphs [103]. We can also identify more easily interactions between/among informational elements and/or tasks. For example, we can simply analyze how do informational elements relate to each other through tasks, how do tasks relate to each other, and even which tasks might be disabled if a resource becomes unavailable.

Note that start (end) conditions are not necessary in a metagraph since any invertex (outvertex) can serve as a potential start (end). Even though we include start and end conditions, we can perform the analysis of a process, where any invertex (outvertex) can serve as a potential start (end). This enables us to represent multiple processes in a single metagraph, which can be useful to get a holistic view of all the processes existing in a single environment/company.

4.2 Verifying policies

In this section, we show how we use the metagraphs to achieve workflow policy verification. When a policy administrator defines the policy to be deployed for a given system, they will first model the behavior of the system at a high level, by writing a policy specification. The administrator then refines the policy specification into a policy implementation (to enforce the policy on the system).

By modeling as metagraphs the high-level policy specification, as well as the translated policy implementation, not only can we look for conflicts and redundancies at both levels [28], but we can also compare the two in order to track deployment errors. If the specification and implementation metagraphs are equal, they then match (the policy implementation corresponds to the policy specification) and no error occurs, but when they do not match, the metagraphs are not equivalent – it means that errors occurred during the refinement and/or deployment. Figure 4.3

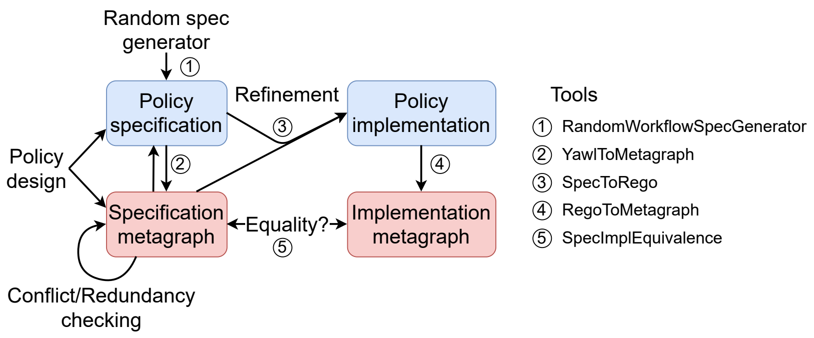


Figure 4.3: Enabling policy verification using metagraphs. When designing the policy, a policy administrator can either choose to create their policy specification by visually representing the policy in the form of a conditional metagraph, or choose to specify the policy in another format. If the specification and implementation metagraphs are equal, we conclude that the policy implementation corresponds to the policy specification. They do not match if the metagraphs are different: we conclude that deployment errors occur. In addition, and by design, the specification metagraph can be used to check for redundancies and conflicts in the policy at both levels (high level specification and ground deployment).

summarizes our overall construction, as well as the tools we developed to achieve verification.

Our idea is generally applicable to any type of policy specification and any kind of policy implementation, provided that both can be converted into metagraphs (e.g UML [117]). We provide support for YAWL specifications, but also support a more general policy format in the form of $\langle source, destination, policy \rangle$ triples.

[117]: Rumbaugh et al. (1999), 'The unified modeling language'

To implement the policies, we consider Rego, a high-level declarative language built for expressing complex policies. Rego is the native query language of the Open Policy Agent (OPA) [27] project. OPA is an open source, general-purpose policy engine, i.e. a system answering policy-related queries according to a policy configuration. OPA is used by companies like Netflix [32], Atlassian, Plex [118] and many others to enforce access control in their release processes and service meshes.

4.2.1 Policy specification into a conditional metagraph – as denoted ② in Figure 4.3

From YAWL to metagraph

To transform this process into a conditional metagraph, we need to define the variables set, the propositions set and the edges set defining the conditional metagraph. To this end, we parse the YAWL file defining the process. A YAWL file is an XML file, from which we can extract relevant information, such as the name of tasks, their inputs and outputs, predicates used for flow control, etc.

The union of elements in the *inputs* and *outputs* of each task make up the *variables set* of the conditional metagraph. The union of predicates of each task make up the *propositions set*. We complete the *edges set* of the metagraph by iterating on all the tasks of the process. We summarize the conditions and actions to take in Figure 4.4. For each task, irrespective of their order in the YAWL process, relevant elements of the YAWL language are identified. The inputs (outputs) of a task correspond to the invertex (outvertex) of the edge, whereas predicates correspond to propositions in the invertex of the edge. The join code indicates the join operation of the currently processed task, and is used to determine the specific actions in each possible case (AND, OR, XOR), which were defined in Section 2.4. The (non-)completion edge refer to the creation of a new edge from the outputs of the previous task to a proposition signifying the current task has (not) been completed, like we explained earlier with disjoint tasks.

Figure 4.2 represents a part of the transformation of our example – the film production process (Figure 4.1) is translated into a conditional metagraph. Edges represent the tasks of the process, where the invertex (outvertex) represents the inputs (outputs) of the task. As explained in the last section, propositions are added (`_completed` suffix) to make sure that we retain the information of the workflow.

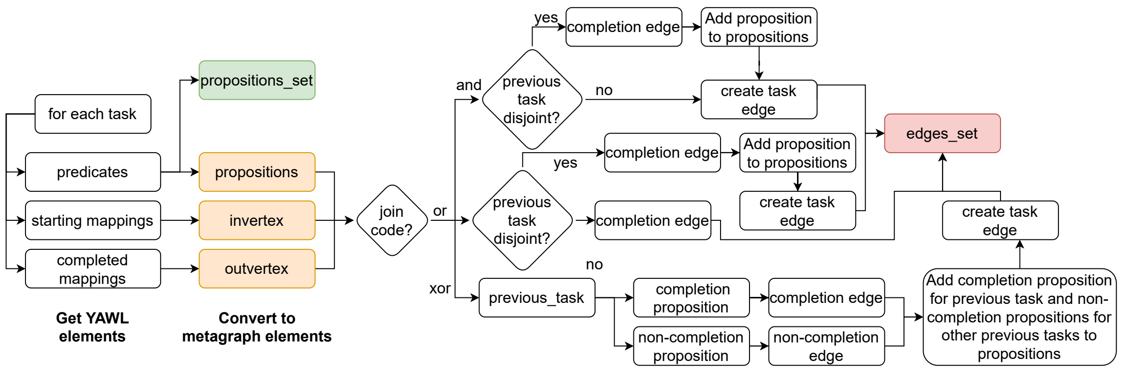


Figure 4.4: Flowchart of transformations from YAWL to a metagraph. For each task, relevant elements of the YAWL language are identified, and converted to metagraph elements. The join code indicates the join operation of the currently processed task, which is then used to determine the actions to perform in each possible case (AND, OR, XOR).

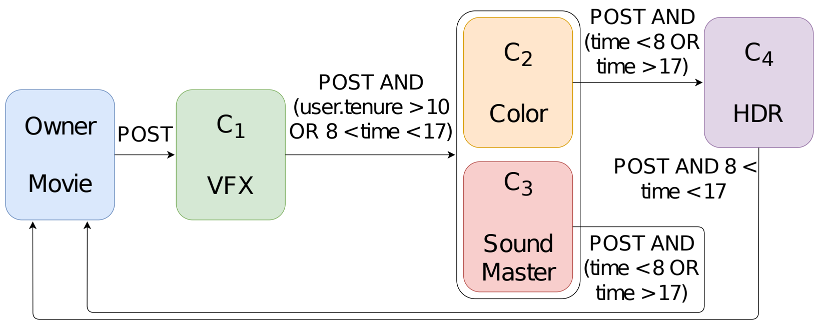

Owner Movie
POST
C1 VFX
POST AND (user.tenure >10 OR 8 <time <17)
C2 Color
C3 Sound Master
POST AND (time <8 OR time >17)
C4 HDR
POST AND 8 < time <17
POST AND (time <8 OR time >17)

Figure 4.5: Movie workflow: special effects apply before color tuning and sound mastering. HDR is set up last.

From a metagraph-like format to metagraph

In addition to the YAWL format which specifies workflows, we have also added the possibility of specifying a workflow directly in a metagraph-like format, which is more generic and applicable to other forms of high-level representations. This form of policy specification can be generically expressed as a list of rules: each describing an edge of the metagraph, as a triplet of the form $\langle source, destination, policy \rangle$.

To transform this kind of policy specification into a conditional metagraph, we also need to define the variables set, the propositions set and the edges set. To this end, we parse the triplets of the policy specification file. A proposition attached to an edge must be true for the edge to be used in a metapath, thus, an `OR` in a proposition can be viewed as separate edges from the same source to the same destination, with each part of the `OR` becoming a sub-proposition attached to one of the newly created edges. Likewise, the `AND` in a proposition means both parts need to be true in order for the edge to be used, so the proposition can't be separated. Those rules will serve as a basis to determine vertices in the metagraph.

In a logical formula, propositions `AND`ed together are part of the same metagraph edge, whereas propositions `OR`ed together are each part of their own metagraph edge. That is the way conditional metagraphs handle connectivity when considering propositions. A proposition attached to an edge must be true for the edge to be used in a metapath, thus, an `OR` in a proposition can be viewed as separate edges from the same source to the same destination, with each part of the `OR` becoming a sub-proposition attached to one of the newly created edges. Considering the proposition between Color and HDR in Figure 4.5, "$POST\ AND\ (time <$

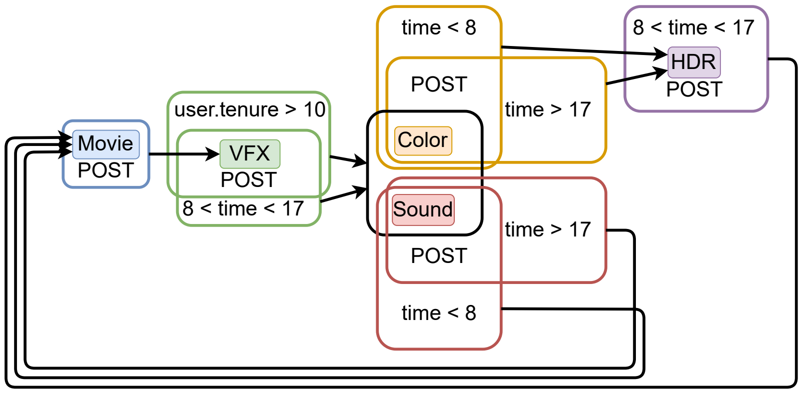


Figure 4.6: Specification metagraph. Elements of the variables set are identified by filled rectangles, and nodes of the metagraph by unfilled rectangles. Propositions being on the edges or in the invertices are equivalent when dealing with conditional metagraphs.

8 $OR\ time > 17$)", we can see the `OR` that separates $time < 8$ and $time > 17$ is responsible for two different edges when we refine the metagraph representation in Figure 4.6. Likewise, the `AND` in a proposition means both parts need to be true in order for the edge to be used, so the proposition can't be separated. Considering again the proposition between Color and HDR in Figure 4.5, we can see the `AND` means that the `POST` component of the proposition is a part of both split edges when we refine the metagraph representation in Figure 4.6.

In order to fill our propositions set as well as the edges set in our conditional metagraph, we need to turn a given logical formula into its Disjunctive Normal Form (DNF). In DNF, a logical formula is composed of `AND`ed propositions `OR`ed together, i.e. smaller logical formulas separated by `OR`s. We can then see that our smaller logical formulas directly correspond to different edges in our metagraph. Take for example the logical formula from Color to HDR in Figure 4.5: $POST\ AND\ (time < 8\ OR\ time > 17)$. We can transform this expression into its DNF, obtaining $(POST\ AND\ time < 8)\ OR\ (POST\ AND\ time > 17)$. The smaller formulas inside the parentheses correspond to the sub-propositions attached to two edges we obtain in our metagraph (Figure 4.6).

Then, since each of our smaller logical formulas are either singular atomic propositions or atomic propositions `AND`ed together, we gather the set of atomic propositions for each edge. Each edge in the conditional metagraph is then generated in the form of a triplet $\langle source, destination, policy \rangle$, where the *policy* component corresponds to one of the smaller logical formulas obtained earlier. We complete the edges set of the metagraph by iterating on all the triplets of the raw specification. The propositions set is simply composed of the union of all atomic propositions, with the generating set being the union of the variables set and the propositions set.

Figure 4.6 represents the transformation of our example: the movie workflow specification (Figure 4.5) is translated into a conditional metagraph. Elements of the variables set are identified by filled rectangles, and nodes of the metagraph by unfilled rectangles. Note that, as suggested in the formal definition of conditional metagraphs, we moved the propositions from the edges of the simple workflow graph to the invertices of the metagraph for a correct and clearer representation on which advanced verification (e.g. checking for conflicts and redundancies) is also possible.

4.2.2 Policy implementation (i.e. Rego) into a conditional metagraph – ④ in Figure 4.3

To transform our policy implementation (i.e. Rego) into a conditional metagraph, we rely on the following approach. We use ANTLR4 [119] , Another Tool for Language Recognition, which is a parser generator used for translating structured files. After constructing our lexer rules and parser grammar for Rego, we were able to generate the Abstract Syntax Tree (AST) for any Rego policy file.

[119]: Parr (2013), *The definitive ANTLR 4 reference*

Listing 4.1 represents the policy for the `Input Shooting Schedule` task in Figure 4.2. As indicated in the metagraph, all `"_completed"` propositions must be true in order for the task to be performed, in addition to resources `production` and `shootingSchedule` being available. Indeed, rules in Rego come in the form $rule\_head\ \{expr_1, expr_2, ..., expr_n\}$. The expressions inside the body of the rule are `AND`ed together when evaluating the rule. All those expressions must be true in order for the rule to be true. Multiple rules defined with the same $rule\_head$ are `OR`ed together – at least one of them must be true for the rule to be true.

Listing 4.1: This is the implementation of the Input_Shooting_Schedule task represented in Figure 4.2.

```
allow {
  input == ["production", "shootingSchedule"]
  output == ["originalTiming", "shootingSchedule", "totalScenes", "totalPageTime"
      ]

  Input\_Cast\_List\_completed
  Input\_Crew\_List\_completed
  Input\_Location\_Notes\_completed
}
```

As expressions `AND`ed together are part of the same metagraph node, whereas propositions `OR`ed together are each part of their own metagraph node, we can perform the transformation of the implementation to the metagraph by looking for this behavior in Rego rules. Thus, by walking the previously obtained AST, one is easily able to generate a metagraph from the policy implementation file. We generated the conditional metagraph for the Rego policy corresponding to Figure 4.1 using ANTLR.

Note that a slight transformation of the implementation metagraph might be necessary. Indeed, the generated implementation metagraph does not have exactly the same structure as the specification metagraph. To finally enable the comparison of metagraphs in this case, we need to apply transformations on the implementation metagraph to make it directly comparable to the specification metagraph. Those operations are defined based on the attributes encountered in the metagraph and dependent of the underlying domain specific language. For example, the `POST` requests in the specification metagraph correspond to the proposition `http_-request.method == "POST"` of the implementation metagraph. In the same way, the implementation metagraph proposition `user_name == "color"` corresponds to an element of the invertex of an edge in the specification metagraph, so it is translated accordingly. Likewise, other propositions and variables of the generating set require transformations, we only need to specify the translation process for each element once (according to the domain specific language). The resulting metagraph should be the same as the one obtained with the specification, provided that the implementation is error-free.

4.2.3 Comparing metagraphs – see ⑤

Once the Rego rules are transformed into an implementation metagraph, we can now compare it to the specification metagraph. To do so, we simply match edges in one metagraph to edges in the other metagraph. The two metagraphs are labeled the same way and are each sorted for efficiency. Once a match is found, we remove, or simply tag, the corresponding respective edges from the two initial set of edges and continue. If both sets are empty (are entirely tagged) after the comparison concludes, the metagraphs match perfectly. The metagraphs are different otherwise, with edges remaining in the sets being the ones that are left unmatched. Those edges are errors/mistakes in the implemented policies, singled out by our comparison. In other words, a non-matching edge indicates an error in the translation process, which can be used to identify which part of the implementation policy has be mistakenly translated or corrupted. Thus, this error identification process pinpoints errors.

In the next section, we measure the performance of our comparison technique to verify the performance of our policy verification method.

4.3 Performance analysis

To profile our policy verification algorithm, we measure the time required to compare the specification and implementation metagraphs.

4.3.1 Methodology

We perform such comparisons by varying different elements, namely the number of elements in the workflow, the number of edges in the workflow, the size of the policy on each edge, and the error rate in the implemented policies. Since our verification method compares edges, we measure the computation time as a function of the edges on the metagraph. Note that the number of propositions on the edges and the number of edges in the workflow determines the number of edges in the resulting metagraphs.

To obtain general and representative results when profiling our algorithm, we chose to generate random workflows (instead of relying on few small real cases). This allows us to get a general idea of how efficient is our algorithm under various conditions. Since the generation is random for most of the variables defining a metagraph, the generated workflows should not exhibit specific structures or policies that may favor or not the comparison. We thus generate random workflows in YAWL.

In practice, we generate the random workflows by varying these sets of parameters:

- **Size of the workflow**, i.e., number of elements in the generating set: 10, 20, 30, 50 or 100. Those correspond to variables which can be used in the input and output of tasks.
- **Policy size**, i.e., number of conditional propositions on each edge for the policy: 2 or 4.

The number of tasks in the workflow, i.e. the number of edges in the metagraph is a multiple of the number of elements in the generating set, as motivated in the following paragraphs.

Had we used simple graphs, we also would have varied the probability of having edges between any two nodes in the graph (i.e. the graph density), as is customary for Erdős-Rényi random graphs. However, the equivalent density property in metagraphs would be to vary the probability of having edges between any two pairs of possible subsets of the generating set: this leads to a combinatorial explosion of the possible number of edges.

This does not sound neither reasonable nor realistic if we compare it with common policy density found in research papers [28, 120] or in GitHub projects. Instead, we use the same (static) ratio between the number of nodes and edges that Ranathunga *et al.* [28] found when they modeled their real-world network policies as a metagraph. They have 1.5 times more edges than the number of elements in the generating set and so do we (to generate our set of workflows). Overall, we generate thirty random conditional metagraphs for each combination of the generation parameters, i.e. the number of elements in the workflow and the policy size, creating 300 different workflow specifications in total (5 generating set sizes, 2 policy sizes, 30 repetitions).

Now that workflow specifications are generated, we need to turn them into their workflow implementation counterparts (i.e. in Rego). As already stated, translating workflow specifications to their real implementations is error prone. We model this by relying on a given percentage of the elements/propositions in the specification randomly changed to another existing element/proposition. To do so, we consider a last parameter:

- **Error rate**, i.e., fraction of errors in propositions of the metagraph. A value of 0.4 means that 40% of the elements/propositions of the metagraph will be changed to erroneous ones; we consider the following error rates: 0.0, 0.2 and 0.4.

We generate errors randomly, resulting in thirty different Rego files for each workflow specification and for each error rate. We obtain ninety different Rego files per workflow specification in total.

Translation is done by iterating over edges of the conditional metagraph generated from the specification (see Section 4.2), which will generate the necessary Rego code. Finally and overall, we obtain 27,000 different policy implementations (300 specifications, 3 error rates, 30 repetitions). The Lines of Code (LoC) of those policy implementations are between 214 and 24729, which is in line with research papers that model real-world policies in terms of LoC size; for example, Ranathunga *et al.* [28] are citing 5043 for one switch configuration in their case study and researchers in [120] are giving an average LoC size between 125 and 1360.

Now that we have the policy implementations, we can translate those into metagraphs to finally perform the comparison. This is achieved using ANTLR and by following the procedure already described in Section 4.2.

Once both specification and implementation metagraphs are generated, we launch our matching algorithm. This algorithm simply compares

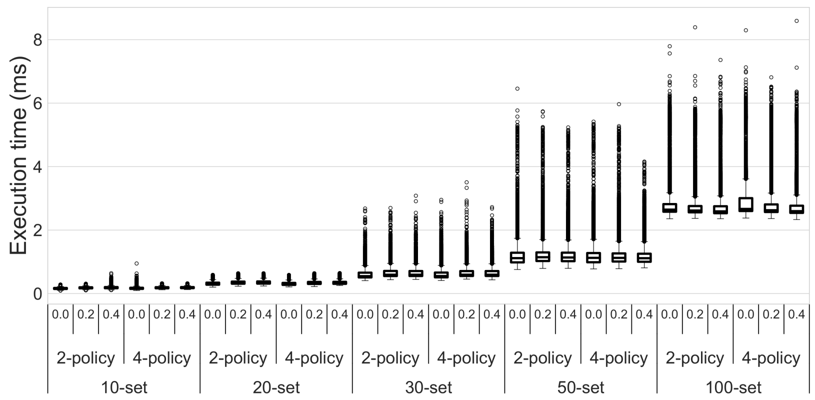


Figure 4.7: Execution time of our matching algorithm according to several different parameters. The error rate has almost no effect on the execution time, while it increases as expected with the number of elements in the generating set.

both metagraphs as two lists of edges. First, for enabling an efficient comparison, we sort both lists by source and destination as respectively the first and second key. Then we try to match edges by iterating through both sorted lists. If both sets are empty after the matching is done, the metagraphs match perfectly. The metagraphs are different otherwise, with the edges remaining in the sets being the ones that are unmatched and the result of errors.

4.3.2 Evaluation

To avoid being subject to the bias effects of peak machine load on the cpu time, for each of the 27,000 scenarios, we measure the cumulative time of both sorting and matching for 30 runs. We then end up with a total of 810,000 measures. We ran our measurements on a laptop, equipped with an Intel Core CPU 3.5-GHz, 16GB of RAM and running Ubuntu 18.04.

Among the thirty repetitions, we have a few extreme values. We checked they are due to peak machine load and consequently removed these outliers (i.e. all values with a Z-score superior to three); that leads to remove 9619 values out of 810000 (1.19%).

Figure 4.7 represents the time required by our algorithm to detect errors according to the set of parameters in use. As we can see, the error rate has a negligible effect on the duration of the algorithm. On the contrary, as anticipated, the execution time increases with the number of elements in the generating set. The number of elements in the generating set increases the number of edges in the metagraph, by the effect of the 1.5 factor applied on edges. This can be verified in a correlation matrix, which indicates a correlation coefficient of 0.945 between the number of edges and the execution time, and a correlation coefficient of 0.004 between the error rate and the execution time.

The effect of the number of edges on the algorithm time is shown more clearly when looking at Figure 4.8. It plots the execution time against the number of edges in the metagraphs. The dots represent the mean value for a given number of edges and the red line represents the ordinary least squares regression for the nonlinear function $y = \alpha + \beta \cdot m \cdot log(m)$. We rely on this function as the average time complexity of our sorting (and matching) algorithm is given by $O(m \cdot log(m))$, with m the number of edges. That is the complexity is dominated by the sorting pre-processing (relying on a binary heap for worst cases or using a quicksort like

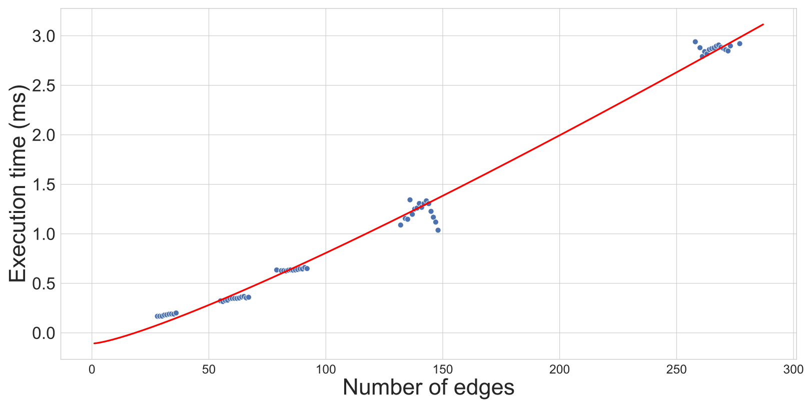


Figure 4.8: Log regression of the execution time according to the number of edges. When the algorithm time is predicted, we found that the number of edges ($\beta = 0.0020$; $p < 0.001$) is a significant predictor.

algorithm for improving average performances). This stage is applied before the actual matching that is then simply linear in the number of edges. When the algorithm time is predicted, we found that the number of edges ($\beta = 0.0020$; $p < 0.001$) is a significant predictor. Indeed, the overall model fit is: $R^2_{adj} = 0.898$, with the *post hoc power analysis* indicating a power greater than .999.

In summary, we can argue that our policy verification method can be efficiently implemented as long as the number of propositions in the policy is reasonable. The complete data, code to generate the measures and results, as well as some guidance are publicly available[1] .

1: See `https://zenodo.org/record/4912289`.

4.4 Discussion

In this chapter, we described how to perform the verification of policies using metagraphs. Using YAWL as a common model of workflows, we showed how metagraphs could accurately represent them by translating YAWL elements to metagraph elements. Thereby, we argue metagraphs are suitable forms of policy modeling. In addition, we state their formal but intuitive and graphical foundations can greatly help reasoning about those policies, and guide policy administrators when designing them. This in turn enables us to maintain more secure workflows. This can prove especially useful in the case of multi-party workflows where networking devices belong to distinct domains and clouds and require complex interactions.

Our thorough performance evaluation has highlighted both the relevance and efficiency of metagraph comparison to verify that deployed policies match their specification in a very reasonable time, even for a large number of rules. To obtain general results, we generated random workflows according to various parameters, and repeated the measures a large number of times. Considering our reduction ($\beta = 0.0020$; $p < 0.001$) and model fit ($R^2_{adj} = 0.898$), as well as our post hoc power analysis ($1 - \beta > 0.999$), we feel confident that our results are representative.

To the best of our knowledge, this is the first time someone has shown how to verify policies using metagraphs. However, some limitations still exist in our verification scheme. First, we assumed in this contribution for the sake of simplicity that both the policy specification and its Rego implementation relied on the same identifiers, although Rego uses some domain specific keywords. This is not always the case, as sometimes the specification and implementation of a policy are not correlated in this way. Some mapping can exist between identifiers in the specification and identifiers in the implementation, but this is not done automatically. One would need to consider the use of pattern matching to realize such a mapping.

Second, even though we are confident our results are representative for the reasons above, we think this could be done better. More specifically, we deplore that there is not any representative policy dataset existing online. To elaborate, most policies we found were either contained in articles, or in the wild in public GitHub repositories. However, the vast majority of papers dealing with policies do not disclose them, as they are either private or currently in use. To the best of our knowledge, there isn't any publicly available policy dataset, which contain policies representative enough to conduct analysis on them. Such a dataset would be of enormous help, as different solutions in the policy verification and analysis literature could then be benchmarked and compared against the same set of relevant policies. We also realize however the difficulty of constructing a policy dataset which would be representative enough for such comparisons to take place. This dataset would in theory need to encompass the full extent of the policy language it used, and also represent policies used in real-life scenarios. Those policies being rarely divulged, and the many ways a policy language can be used, makes it difficult to build this dataset.

4.5 Conclusion

In this chapter, we detailed to what extent metagraphs are the most appropriate structures to naturally model workflow policies. We showed we can fully express YAWL with metagraphs, and how each task, condition and operator can be converted into a metagraph representation. We therefore argue the richness of the YAWL language is supported by metagraphs. We then showed how to use metagraphs for workflow policy verification to pinpoint errors in a policy. In particular, we introduced a method to verify YAWL specifications. For this, we relied on a policy implementation based on Rego, a high-level declarative language built for expressing complex policies. We finally evaluate the performance of our verification method, and find that we can check policies in a very reasonable time, even when the policy is large.

This work has led to two publications [20, 21], the development of a Python 3 version of MGToolkit and the development of a policy verification framework. Those tools are publicly available.

Overall, we highly recommend the use of metagraphs to represent, verify and analyze workflow and security policies alike. Possible future works include broadening the types of policies our verificator can handle. We can currently deal with workflow and security policies, but assume both specification and implementation use the same identifiers. To extend the domain of policies we are able to verify, a possible lead is to investigate pattern matching solutions in order to provide a more general way of interpreting identifiers. Namely, we could rely on the work done by Clifton *et al.* [121] or Kim *et al.* [122] in this area.

Analyzing Policies to Find Redundancies 5

Chapter 3 allowed us to deploy a workflow securely using the microservices infrastructure. We can also check if the implementation of the policy is correctly enforced. Chapter 4 then explained how potential shortcomings in the policy could be avoided, by comparing its specification to its implementation. Therefore, we can be sure that the policy specification matches its implementation, and that it is then enforced correctly.

However, a problem arises if the initial specification itself is incorrect. The policy specification must represent the intent of the policy administrator. Making sure the intent of the policy administrator is appropriately translated into the specification is hard. Indeed, a policy specification can contain conflicts and redundancies.

Nevertheless, we can take some measures to acquire guarantees over the policy specification, to make sure it contains no conflicts or redundancies. In this chapter, we are specifically concerned with redundancies in the policy.

Besides correctness, one might want to remove redundancies for performance reasons, since redundancies can still be evaluated by a policy engine, but never impact the final access control decision. Another advantage is that it removes clutter from policy specifications. Policies can get quite large and messy, so any removed redundancy helps clear the policy and eases its management. As a lot of data exposures are caused by human error, having a clearer policy will surely help policy administrators to make less errors when handling them. All those factors point to the elimination of redundancies as a way to improve a sub-optimal policy.

We propose to use metagraphs in order to achieve this goal of policy analysis. As in Chapter 4, metagraphs are a good representation for policies, and can also be used in the case of analysis. Ranathunga *et al.* [28] have already described a procedure using metagraphs to find both conflicts and redundancies inside a policy, but their method has shortcomings, as further detailed in Section 5.2.

Note that we do not treat conflicts here. Ranathunga *et al.* [28] find conflicts by keeping a list of properties conflicting with each other. Then, they look at properties contained in a metapath. If conflicting properties appear in the same metapath, they apply domain-specific knowledge to determine if the conflict is valid. This approach works great when the number of metapaths is limited, but suffers from the same shortcomings as the method to detect redundancies, as we will see next.

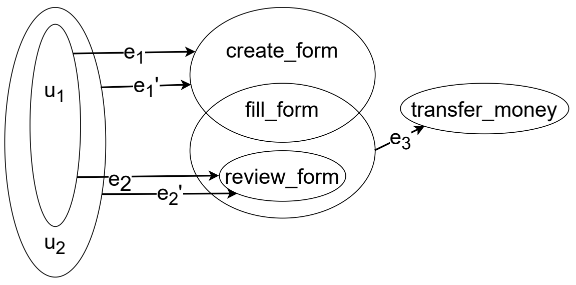


Figure 5.1: A metagraph.

5.1 An approach to policy analysis: problem statement

We focus specifically on the identification and removal of redundancies. Modeling a policy as a metagraph helps us identify redundancies. Consider a policy with a set of users and a set of resources. When modeled as a metagraph, permissions of users to access resources are translated as metapaths. Moreover, dominant metapaths from the users to the resources correspond to the minimal set of elements in order to achieve connectivity, i.e. to model the policy. Therefore, all elements that are not on a dominant metapath correspond to redundancies in the policy.

Note that the notion of redundancy is always in reference to a specific point of view. In this case, redundancies are in reference to the users that want to access resources. We need this point of reference, otherwise the notion of redundancy would be restricted to cases like policy rules appearing twice. This restricted definition is much less useful, and those cases can usually be solved with a simpler method. This can be illustrated quite clearly in a metagraph. Since every edge is a dominant metapath from its invertex to its outvertex, no element of the metagraph can be removed.

Consider Figure 5.1, which represents a metagraph. The edge e_1 is a dominant metapath from its invertex, $\{u_1\}$, to its outvertex, $\{create\_form, fill\_form\}$. This means e_1 can never be removed. We can say the same thing about every edge in the metagraph, so we need some context to discriminate elements into the relevant and redundant categories.

For example, if we consider redundancies in the context of user $\{u_1\}$ wanting to access $\{transfer\_money\}$, suddenly edges e_1' and e_2' are redundant, since the metapath $M_1(\{u_1, u_2\}, \{transfer\_money\}) = \langle e_1', e_2', e_3 \rangle$ is not dominant. Indeed, the metapath $M_2(\{u_1\}, \{transfer\_money\}) = \langle e_1, e_2, e_3 \rangle$ is a metapath and has a subset of the input of M_1. We can thus remove edges e_1' and e_2'.

More generally, we want to remove redundancies with regard to a set of source/destination couples. Those couples represent elements of interest for the policy administrator, with one of its objectives when analyzing the policy being making sure connectivity between those elements is not redundant, while preserving said connectivity.

To find those redundant elements, we cannot just consider any ordered pair of sets of elements as source and target for a metapath, since an element can be redundant with regard to a certain source/target pair, but not to another.

Without losing generality, we can reduce the problem to finding redundancies for a single source/target pair. Finding redundancies for multiple pairs is then simple, since we only need to apply the solution for a single pair to all pairs, and eliminate common redundant elements.

A straightforward method to find redundant elements is to find all dominant metapaths between the source and the target. The union of elements of these metapaths, i.e. the union of all variables in the vertices, the union of all propositions in the vertices as well as the union of edges, can then be compared respectively to the variables set, the propositions set and the edges set used to construct the metagraph. Elements present in the sets used to construct the metagraph but not present in the union of elements of the dominant metapaths can be removed, as they are not part of any dominant metapath. This method can be generalized to multiple source/destination couples by considering the union of all dominant metapaths of those couples.

Looking again at Figure 5.1, we can determine that in this simple metagraph, the only dominant metapath from $\{u_1\}$ to $\{transfer\_money\}$ is $M(\{u_1\}, \{transfer\_money\}) = \langle e_1, e_2, e_3 \rangle$. If we consider the union of elements and edges used in the metapath, we get $\{u_1, create\_form, fill\_form, review\_form, transfer\_money\}$ for the elements and $\{e_1, e_2, e_3\}$ for the edges. When comparing this to the variables set and edges set of the metagraph, the remaining elements are u_2, e'_1 and e'_2. Those elements can thus be deleted.

Ranathunga *et al.* [28] , the only ones in the literature describing how to use metagraphs to detect redundancies, also use metapaths. Unfortunately, shortcomings of their method render it unusable in practice.

[28]: Ranathunga et al. (2020), 'Verifiable Policy-Defined Networking using Metagraphs'

5.2 Shortcomings of the currently existing method

Similarly to the straightforward method described above, Ranathunga *et al.* [28] make use of metapaths to find redundancies. The algorithm they use can be found in Algorithm 1.

They advise to "...simply check all feasible metapaths in a policy metagraph for edge and input dominance, if either fails, the policy includes redundancies". This however poses three main issues, which we go over in more detail below.

Computing $A*$ takes too long.

To check all feasible metapaths in a policy metagraph for edge and input dominance, Ranathunga *et al.* first need to find all feasible metapaths. To do so, they make use of the procedure described in the Basu and Blanning book [103] .

[103]: Basu et al. (2007), *Metagraphs and their applications*

This procedure is based on the computation of the transitive closure of the adjacency matrix of the metagraph, $A*$. As a reminder, $A* = A + A^2 + \ldots$ represents all simple paths of any length in the metagraph. The complexity of computing the $A*$ matrix is thus $O(n^3)^m$, with n the number of elements

Algorithm 1 Detect policy redundancies and conflicts using Metagraph algebras [28]. The policy metagraph is *pmg*.

```
procedure DetectPolicyInconsistencies(pmg)
    $R_1 \leftarrow$ dict()
    $R_2 \leftarrow$ dict()
    $processed \leftarrow []$
    $redundancies \leftarrow []$
    $conflicts \leftarrow []$
    for all $e_1 \in$ pmg.edges do
        for all $e_2 \in$ pmg.edges do
            if $e_1 \neq e_2$ and $(e_2, e_1) \notin processed$ then
                $i_1 \leftarrow e_1$.invertex $\cap$ $e_2$.invertex
                $i_2 \leftarrow e_1$.outvertex $\cap$ $e_2$.outvertex
                $i_3 \leftarrow e_1$.propositions $\cap$ $e_2$.propositions
                if len($i_1$) > 0 and len($i_3$) > 0 then $R_1[e_1]$.append($e_2$)
                if len($i_2$) > 0 and len($i_3$) > 0 then $R_2[e_1]$.append($e_2$)
            $processed$.append(($e_1, e_2$))
    for all $e_1, v_1 \in R_1$ do
        $src \leftarrow e_1$.invertex
        for all $e_2 \in R_1[e_1]$ do
            $src \leftarrow src \cup e_2$.invertex
        for all $e_3, v_3 \in R_2$ do
            $target \leftarrow e_3$.invertex
            $i_4 \leftarrow e_1$.propositions $\cap$ $e_3$.propositions
            if len($i_4$) > 0 then
                $mps \leftarrow$ GetMetapaths($src, target$)
                for all $mp \in mps$ do
                    if not IsDominantMetapath($mp$) then
                        $redundancies$.append($mp$)
                    if PropositionsConflict($mp$) and
                            IsValidConflict($mp$) then
                        $conflicts$.append($mp$)
    return (redundancies, conflicts)
```

in the metagraph and m the length of the longest path in the metagraph. This is a major issue, since the computation of this matrix is feasible only for very small metagraphs, and infeasible for most.

Only some redundancies are detected

Ranathunga *et al.* propose to filter potential sources to try alleviating some of the computational load of the entire method. They state that only overlapping invertices[1] can cause redundancies. They get faster results, but they only detect some of the redundancies. In this case, note that redundancies are in reference to metapaths starting with overlapping vertices.

1: In other words, edges whose invertices share elements.

This is a concern as well, since the method might not be efficient. Note that even if they reduce the load, the computation of the $A*$ matrix is still required beforehand.

A manual check is required

Let us assume we find all feasible metapaths anyway, and check them for edge and input dominance. Once those redundancies are found, a check with the policy administrator is still necessary to fix the policy.

Here, the reference frame of redundancies here is metapaths starting with overlapping vertices instead of actual users and resources or other contextualising elements. As such, the policy administrator needs to approve the deletion of redundancies beforehand, in the case elements found by the procedure are indeed redundant. An automatic redundancy deletion method is therefore desired, since the redundancies found can sometimes be difficult to check, and the policy administrator can make some mistakes.

Computation times in practice

Using the method described by Ranathunga *et al.*, it took a full hour to process metagraphs of 13 elements at most, and we missed most of the redundancies. As a comparison, we implemented the algorithm enumerating all metapaths, which took a full hour to process metagraphs of 7 elements at most, but gave us all redundancies.

We therefore need to find another way to identify redundancies. Instead of trying to build all dominant metapaths, another approach would be to determine, for each edge, if it belongs to any dominant metapath or not. If it does, the edge is not a redundancy, otherwise it is. We prove in the next section that this problem is NP-Hard.

5.3 Proof of NP-Hardness

Since we could not find any algorithm that can identify redundancies in polynomial time, we made sure there exists no such algorithm. Before describing our efficient exhaustive algorithm to find redundancies, we need a proof of complexity.

This is where hypergraphs come into play. Like it was explained in Chapter 2, metagraphs and hypergraphs are both graph theoretic structures that generalize the classical graph. Here, we exploit the fact that a problem fundamental to hypergraphs, finding hypernetworks, is related to our problem of finding redundancies in a metagraph.

Although metagraphs and hypergraphs have been studied separately, they are similar. Before describing how finding hypernetworks is related to the problem of finding redundancies, we explicit the relation between metagraphs and hypergraphs. Note that we are concerned with directed hypergraphs, and thus will refer to directed hypergraphs simply as hypergraphs.

5.3.1 Metagraphs and Hypergraphs: a comparison

Here, we point out the similarities and differences between metagraphs and hypergraphs. Where the two differ, we integrate concepts from metagraphs into hypergraphs to create a more general and expressive hypergraph definition that includes metagraphs.

Vertices and edges

Metagraphs and hypergraphs are based on relationships between subsets of elements. In metagraphs we call the universe of elements the generating set. In hypergraphs we call it the vertex set. The relationships in metagraphs are called edges. In hypergraphs, we have hyperedges, which can be B-edges and F-edges.

All these concepts have clear equivalencies. The generating set and vertex set are semantically equivalent. A metagraph edge describes a hyperedge. Note that we use vertex and hypervertex when describing a hypergraph interchangeably. Likewise, edge and hyperedge are used interchangeably.

Metapaths and hyperpaths

It is natural to assume that metapaths and hyperpaths are natural analogs. For the most part, this is true. However, there are some subtle differences which we seek to resolve.

The definition of a metapath (Definition 2.4.2) does not stipulate an ordering on the edges comprising the metapath. However, the definition of a hyperpath does (Definition 2.4.10). These parts of the definition are actually inconsequential. The original definition of a hyperpath given by Gallo *et al.* does not stipulate an edge order. Any unordered hyperpath can be given an ordering (using the visit order from [104]), and any ordered hyperpath can be converted to an unordered set of edges.

[104]: Gallo et al. (1993), 'Directed hypergraphs and applications'

We suggest that the terms *ordered hyperpath* and *unordered hyperpath* be used to refer to these two variants. Sometimes an ordering is preferred, since it gives a more path-like sequence of steps to follow. As such, when we say hyperpath, assume we mean an ordered hyperpath.

Metapaths go from a *source* set of elements to a *target* set of elements. Hyperpaths go from a source vertex s to a target vertex t. Gallo *et al.* [104] note that set-to-set hyperpaths can be modelled by adding a new F-edge and B-edge for the source and target sets respectively. We recall that hyperpaths are defined as such:

Definition 5.3.1 (Hyperpath) *Let $\mathcal{H} = (V, E)$ be a directed hypergraph and let $s, t \in V(H)$. A hyperpath from s to t in $\mathcal{H}$ is a minimal subhypergraph (where minimality is with respect to deletion of vertices and edges) $\mathcal{H}' \subseteq \mathcal{H}$. Further, the hyperedges comprising $\mathcal{H}'$ can be ordered in a sequence $\langle e_1, e_2, \ldots, e_k\rangle$ such that for every edge $e_i \in \mathcal{H}$ it is the case that $T(e_i) \subseteq \{s\} \cup H(e_1) \cup H(e_2) \cup \cdots \cup H(e_{i-1})$ and $t \in H(e_k)$.*

We give a modified version of Definition 2.4.10 to work with a source set of vertices $\mathcal{S}$ and target set of vertices $\mathcal{T}$:

Definition 5.3.2 (Hyperpath Between Sets) *Let $\mathcal{H} = (V, E)$ be a directed hypergraph and let $\mathcal{S}, \mathcal{T} \subseteq V(H)$. A hyperpath from $\mathcal{S}$ to $\mathcal{T}$ in $\mathcal{H}$ is a minimal subhypergraph (where minimality is with respect to deletion of vertices and edges) $\mathcal{H}' \subseteq \mathcal{H}$. Further, the hyperedges comprising $\mathcal{H}'$ can be ordered in a sequence $\langle e_1, e_2, \ldots, e_k \rangle$ such that for every edge $e_i \in \mathcal{H}'$ it is the case that $T(e_i) \subseteq \mathcal{S} \cup H(e_1) \cup H(e_2) \cup \cdots \cup H(e_{i-1})$ and $\mathcal{T} \subseteq \mathcal{S} \cup H(e_1) \cup H(e_2) \cup \cdots \cup H(e_k)$.*

Input dominance and edge dominance

A metapath can be input-dominant (Definition 2.4.4) or edge-dominant (Definition 2.4.5). These mean minimal with respect to source vertex deletion (input dominance) or edge deletion (edge dominance).

These concepts have never been defined for hyperpaths. We observe that existing definitions, including the ones we give, enforce that a hyperpath be edge-dominant, as it must be a minimal subhypergraph. Metapaths, in contrary, do not need to be edge-dominant. To introduce this concept to hypergraphs, we define an *edge-redundant hyperpath* as a hyperpath that is not minimal with respect to edge deletion. Hyperpaths being edge-dominant is an important property we use in our proofs.

Input dominance does not make sense for the vertex-to-vertex definition of hyperpaths, since they proceed from a single source vertex. However, under Definition 5.3.2 the idea of input dominance can be adopted from metagraphs. We define an *input-dominant hyperpath* as a hyperpath from a source set $\mathcal{S}$ to a target set $\mathcal{T}$ if there exists no hyperpath from $\mathcal{S}' \subset \mathcal{S}$ to $\mathcal{T}$.

5.3.2 Problem definitions and relations

Redundancy and forced edges

We now explain how hypernetworks relate to the problem of finding redundancies. We first define our problem, which is to find all redundant edges in a hypergraph, in definition 5.3.3.

Definition 5.3.3 (Redundant Hypergraph Edge Problem) *An edge e in a hypergraph $\mathcal{H} = (V, E)$ (or equivalently a metagraph) is redundant with respect to a given source set $\mathcal{S} \subseteq V$ and target set $\mathcal{T} \subseteq V$ if there is no input-dominant hyperpath $\mathcal{H}'$ from $\mathcal{S}$ to $\mathcal{T}$ such that $e \in \mathcal{H}'$. The* Redundant Hypergraph Edge Problem *(RHEP) is to find all redundant edges given $\mathcal{H}$, $\mathcal{S}$, and $\mathcal{T}$.*

The Redundant Hypergraph Edge Problem (RHEP) is equivalent with respect to computational complexity up to polynomial factors to the Forced Hyperpath Edge Problem (FHEP), which we define below.

Definition 5.3.4 (Forced Hyperpath Edge Problem) *The* Forced Hyperpath Edge Problem *(FHEP) is a decision problem in which we decide if there exists any input-dominant hyperpath $\mathcal{H}'$ in a hypergraph $\mathcal{H} = (V, E)$ between a source set $\mathcal{S}$ and a target set $\mathcal{T}$ that must contain an edge $e \in E$.*

We informally prove the equivalence of the RHEP and FHEP by doing a reduction in either direction such that the reduction requires only polynomial time.

The RHEP can be reduced to the FHEP by observing that we can try forcing each possible edge in the hypergraph by calling a solution to the FHEP. Any edge that cannot be forced is redundant.

The FHEP can be reduced to the RHEP by observing that any redundant edge cannot be forced, and any non-redundant edge can be forced. As such, we can say that the FHEP is the decision version of the RHEP problem.

Next, we show that the FHEP is equivalent to the problem of computing an (s, d)-hypernetwork (Figure 5.2).

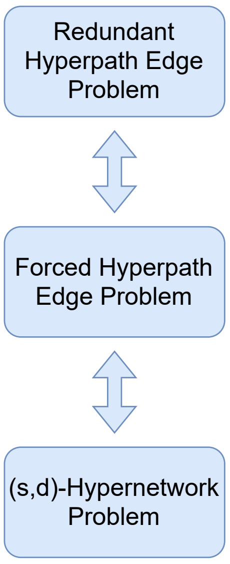

Redundant Hyperpath Edge Problem
Forced Hyperpath Edge Problem
(s,d)-Hypernetwork Problem

Figure 5.2: The RHEP is equivalent to the FHEP, which is equivalent to the SDHP.

Forced edges and (s,d)-hypernetworks

Definition 5.3.5 ((s, d)-Hypernetwork Problem) *The* (s, d)-Hypernetwork Problem *(SDHP) is to find the (s, d)-hypernetwork $\mathcal{H}_{s,d}$ given a hypergraph $\mathcal{H}$.*

We also informally prove the equivalence of the FHEP and SDHP by doing a reduction in either direction such that the reduction requires only polynomial time.

We can reduce any FHEP instance to a SDHP instance in polynomial time. The FHEP is defined between a source s and destination d on a hypergraph $\mathcal{H}$. Suppose we could solve the SDHP between s and d on $\mathcal{H}$. Any edge in the (s, d)-hypernetwork is forcible in the FHEP sense because there must be a hyperpath using it. Conversely, any edge not in the (s, d)-hypernetwork cannot be forcible in the FHEP sense because there must be no hyperpath using it. Thus, the FHEP is reducible to the SDHP.

In the same way, we can reduce any SDHP instance to a instance in polynomial time. The SDHP is defined between a source s and destination d on a hypergraph $\mathcal{H} = (V, E)$. Suppose we could solve the FHEP between s and d on $\mathcal{H}$, for each edge $e \in E$. Any edge that can be forced will be in the (s, d)-hypernetwork since there is a hyperpath using it. Likewise, any edge that cannot be forced will not be in the (s, d)-hypernetwork because there is no hyperpath using it.

In summary, the RHEP is equivalent to the FHEP, which is itself equivalent to the SDHP.

This is useful to us, since existing complexity results on the SDHP can help us unveil more about the complexity of the FHEP and RHEP. Volpentesta [4] describes polynomial algorithms for finding s-hypernetworks, and more importantly for finding (s, d)-hypernetworks in acyclic B-hypergraphs. Later, Pretolani [123] clarified the findings and terminology

[4]: Volpentesta (2008), 'Hypernetworks in a directed hypergraph'

[123]: Pretolani (2013), 'Finding hypernetworks in directed hypergraphs'

of Volpentesta and provided a linear time solution to finding (s, d)-hypernetworks in acyclic B-hypergraphs.

Here, we give a table of current complexity classes for finding a hyperpath in the literature. The found hyperpath can either be normal (regular) or with a forced edge. The hypergraph in which we need to find a hyperpath can either be an B, F, or BF-hypergraph, and can be cyclic or acyclic. Finally, the found hyperpath either requires edge-dominance[2] , input dominance[3] or both.

2: A hyperpath is minimal, so edge-dominant.

3: In this case, edge dominance does not hold so it can be an edge-redundant hyperpath.

Table 5.1: Complexity summary for finding a hyperpath.

			Edge-dom	Input-dom	Dom
Regular	Cyclic	B	P (linear)*	P (linear)†	P‡
		F	P*	P†	P‡
		BF	P*	P†	P‡
	Acyclic	B	P (linear)*	P†	P‡
		F	P*	P†	P‡
		BF	P*	P†	P‡
Forced Edge	Cyclic	B	NP-Hard [4]	?	?
		F	NP-Hard [4]	?	?
		BF	NP-Hard [4]	?	?
	Acyclic	B	P (linear) [4]	?	?
		F	?	?	?
		BF	?	?	?

*: Find the s-hypernetwork, prune edges.

†: Find the s-hypernetwork. Repeat with a subset of the input until done.

‡: Find the s-hypernetwork. Repeat with a subset of the input until done. Prune edges.

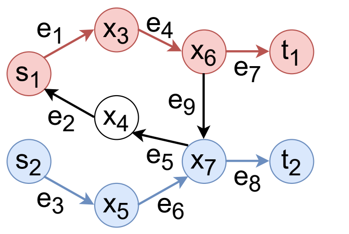
Note that computing an (s, d)-hypernetwork is NP-hard if cycles are permitted. Volpentesta hints that the problem in the general case can be reduced to the well known problem of finding two pairwise disjoint paths between two pairs of nodes [4]. Indeed, since the 2-disjoint paths problem is NP-complete [124], the problem of deciding whether a given edge belongs to the (s, d)-network in a directed graph is NP-complete as well. However, they do not give the proof of the reduction in the paper, so we give it in the next section.

5.4 Finding (s,d)-hypernetworks in the general case

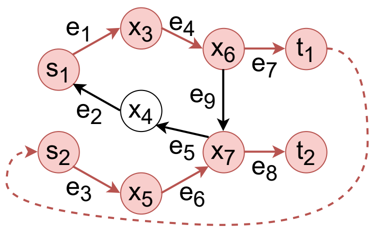
We show that the FHEP in B, F, and BF-hypergraph is NP-Complete even if input dominance is relaxed. Our proof is by reduction from an analogous problem in classical graphs, which hypergraphs naturally generalize.

Let us formulate a classical directed graph version of the FHEP. A path in a graph is a sequence of one or more vertices such that consecutive vertices in the path are connected by edges. A simple path in a graph is a path containing no repeated vertices or edges. Suppose we have a source vertex s and a target vertex t. The Forced Path Edge Problem (FPEP) is to decide if there exists a simple path from s to t using a forced edge e.

To prove the FPEP is NP-Complete, we use a reduction from the 2-Vertex Disjoint Path Problem (2-VDPP) which is known to be NP-Hard [125]. The 2-VDPP problem consists of deciding if there exist two vertex disjoint



(a) A graph G with two disjoint paths $p(s_1, t_1)$ and $p(s_2, t_2)$.



(b) The corresponding graph construction G', a path $p(s_1, t_2)$ with a single forced edge (t_1, s_2).

Figure 5.3: An example of our 2-VDPP construction.

simple paths, one from vertex s_1 to vertex t_1 and the other from s_2 to t_2.

Suppose we have an instance of a 2-VDPP problem on a digraph $G = (V, E)$ with a vertex set V and an edge set E. Let $s_1, t_1, s_2, t_2 \in V$ be the terminals. In our construction, we create a modified version of G called G' in which we add an edge $e' = (t_1, s_2)$ from t_1 to s_2 if it does not already exist in G.

Figure 5.3 shows an instance of the 2-VDPP problem on a graph G, and the corresponding construction G'.

Lemma 5.4.1 *A simple path from s_1 to t_2 that forces the edge e' in the construction G' represents a solution to the corresponding 2-VDPP problem.*

Proof. We can prove Theorem 5.4.1 by choosing e' as the single forced edge. A solution to a FPEP is a simple path from s_1 to t_2 via e'. Such a path in G' could be split into two parts, the part from s_1 up to and including t_1, then the part from s_2 to t_2. These two parts must be vertex disjoint, or the path would not be a simple path. Further, we could delete e' and the two parts would become two simple paths that correspond to a solution to the 2-VDPP instance. □

Lemma 5.4.2 *The forced path edge problem is NP-Complete.*

Proof. This follows from Lemma 5.4.1. By reduction, the FPEP must be at least as hard as the 2-VDPP. □

Theorem 5.4.3 *The forced hyperpath edge problem is NP-Complete.*

Proof. Any classical graph G can be embedded in a hypergraph $\mathcal{H}$. Each vertex in G can be represented by a vertex in $\mathcal{H}$. Each edge in G can be represented by a hypergraph edge between singleton sets in $\mathcal{H}$. A hyperpath in $\mathcal{H}$ corresponds to a simple path in G.

Because hypergraphs (B, F, and BF) generalize classical graphs this way, a reduction from the FPEP to the FHEP can be achieved by embedding the FPEP in a hypergraph. Since the FPEP is NP-Complete (Lemma 5.4.2), so must the FHEP. □

Computing the (s, d)-hypernetwork is thus possible in linear time in acyclic B-hypergraphs, and NP-Hard in the general case. The remaining case to explore is acyclic F-hypergraphs, which we do in the next section.

5.5 Finding (s,d)-hypernetworks in acyclic F-hypergraphs

Here, we explore the complexity of finding (s, d)-hypernetworks in acyclic F-hypergraphs. It may surprise the reader to see that we prove the problem becomes NP-hard, despite that fact that F-hypergraphs are symmetric to B-hypergraphs.

In the previous section, we showed that the FHEP is reducible to the SDHP. This means that if we can prove the FHEP is NP-complete, the SDHP is therefore NP-hard.

Theorem 5.5.1 *If the FHEP is NP-complete, then the SDHP is NP-hard.*

We prove the FHEP is NP-complete.

Our proof involves a reduction from 3-SAT, which is the Boolean satisfiability problem restricted to exactly 3 variables per clause. It is widely known to be NP-complete [126] .

[126]: Cook (1971), 'The complexity of theorem-proving procedures'

The reduction is achieved by taking an instance of the 3-SAT problem and constructing a corresponding acyclic F-hypergraph such that any found hyperpath with a forced edge would imply a solution to the 3-SAT problem.

Our acyclic F-hypergraph construction

Assume a 3-SAT instance with variables $v_1, \dots, v_n$ and clauses $c_1, \dots, c_m$. Our corresponding acyclic F-hypergraph construction contains a vertex and a pair of edges for each variable, and three vertices and three edges for each clause.

We start with an initial vertex p_0 which is the source of our hyperpath. We also create a vertex q_0 that will be used to force the single must-use edge of our hyperpath, and a node f which is the target of the hyperpath we need to find. For each variable v_i, we create a node p_i. Then for each clause c_i, we create three nodes $q_{i,1}, q_{i,2}, q_{i,3}$ which correspond respectively to the three literals of each clause.

A variable v_i appears in its positive form in a set of a clauses $x_1, x_2, \dots x_a$. For each such clause, v_i appears as either the first, second, or third literal, denoted by $y_1 \in \{1, 2, 3\}, y_2 \in \{1, 2, 3\}, \dots, y_a \in \{1, 2, 3\}$. A variable v_i appears in negated form $(\neg v_i)$ in b clauses. Call the corresponding clauses and literals $x'_1, x'_2, \dots x'_b$ and $y'_1 \in \{1, 2, 3\}, y'_2 \in \{1, 2, 3\}, \dots, y'_b \in \{1, 2, 3\}$ respectively.

In our construction, for each variable v_i we have a vertex p_i and two hypergraph edges. The first edge is of the form $(\{p_{i-1}\}, \{p_i, q_{x_1,y_1}, q_{x_2,y_2}, \dots, q_{x_a,y_a}\})$, and it corresponds to assigning variable v_i to false, thus blocking the clauses in the edge. The second edge is of the form

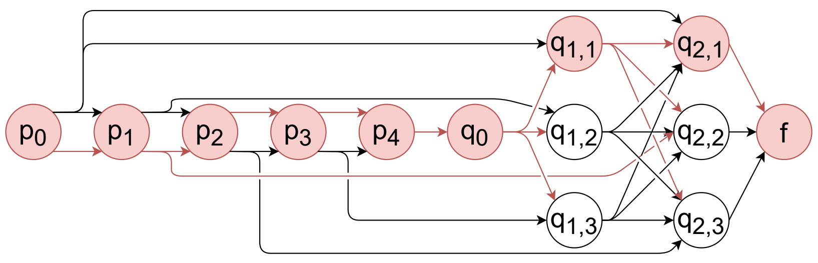


Figure 5.4: An example of our acyclic F-hypergraph construction, corresponding to the following 3-SAT instance: $(v_1 \vee v_2 \vee \neg v_4) \wedge (v_1 \vee \neg v_2 \vee \neg v_3)$. Red hyperedges and nodes indicate a valid hyperpath with the forced edge $(\{p_4\}, \{q_0\})$.

$(\{p_{i-1}\}, \{p_i, q_{x'_1,y'_1}, q_{x'_2,y'_2}, \ldots, q_{x'_b,y'_b}\})$, and it corresponds to assigning variable v_i to true.

For each clause c_i we construct three vertices $q_{i,1}, q_{i,2}, q_{i,3}$ and three edges. Each of the three edges corresponds to one of the three vertices. The j-th such edge has the form $(\{q_{i,j}\}, \{q_{i+1,1}, q_{i+1,2}, q_{i+1,3}\})$. Since c_m is the last clause we construct it differently. We have $(\{q_{m,j}\}, \{f\})$ for $j \in \{1, 2, 3\}$.

Now, we must connect the p vertices to the q vertices. This is done by adding edges $(\{p_n\}, \{q_0\})$ and $(\{q_0\}, \{q_{1,1}, q_{1,2}, q_{1,3}\})$. Note that $(\{p_n\}, \{q_0\})$ is particularly important since it is the forced edge. We aim to find a hyperpath from p_0 to f that must include $(\{p_n\}, \{q_0\})$.

To illustrate our construction, we consider the following 3-SAT instance: $(v_1 \vee v_2 \vee \neg v_4) \wedge (v_1 \vee \neg v_2 \vee \neg v_3)$. The F-hypergraph construction stemming from this instance can be found in Figure 5.4. This instance is composed of four variables, v_1, v_2, v_3, v_4 and two clauses, which correspond respectively to vertices p_1, p_2, p_3, p_4 for the variables, vertices $q_{1,1}, q_{1,2}, q_{1,3}$ for clause 1, and vertices $q_{2,1}, q_{2,2}, q_{2,3}$ for clause 2.

Let us call our hypergraph construction $C(\phi)$ for a 3-SAT instance ϕ. To prove that finding a hyperpath in $C(\phi)$ with the forced edge $(\{p_n\}, \{q_0\})$ is equivalent to a valid variable assignment in ϕ we must first prove some lemmas.

Lemma 5.5.2 $C(\phi)$ *is an acyclic F-hypergraph.*

Proof. Each edge in $C(\phi)$ has a tail containing a single vertex and is therefore an F-edge. It follows from the definition of F-hypergraph that $C(\phi)$ is an F-hypergraph as it contains only F-edges.

Each edge in $C(\phi)$ progresses from the previous layer to the next. Therefore it is acyclic. □

Lemma 5.5.3 *A hyperpath $\mathcal{H}'$ from s to d in an F-hypergraph $\mathcal{H}$ contains exactly one path $P_{s,d}$.*

Proof. We say that a path is contained in a hyperpath if the path is comprised of only vertices and edges in the hyperpath.

There is only one edge e in $\mathcal{H}'$ such that $d \in H(e)$, because a hyperpath is minimal. Since $\mathcal{H}$ is an F-hypergraph, the tail of e contains a single vertex $T(e) = \{v\}$. We can apply the same reasoning to v recursively. That is, remove d and e from $\mathcal{H}'$ and treat v as the target, then repeat. Eventually, we will remove all the vertices from $\mathcal{H}'$ except s. The vertices and edges we removed must be the only path from s to d. □

Lemma 5.5.4 *A hyperpath $\mathcal{H}$ in $C(\phi)$ that uses the edge $(\{p_n\}, \{q_0\})$ will contain contain exactly one edge $e_{v_i} \in \mathcal{H}$ for each variable in $v_i \in \phi$ and exactly one edge $e_{c_i} \in \mathcal{H}$ for each clause $c_i \in \phi$.*

Proof. Lemma 5.5.3 implies that $\mathcal{H}$ contains a single path from p_0 to f. Since we force the edge $(\{p_n\}, \{q_0\})$, there must also be a single path from p_0 to q_0. The only way to get to q_0 is through $p_0 \dots p_n$. For any p_i where $i < n$ there are two edges to choose from, and we must pick exactly one of them on the path to q_0 because hyperpaths are minimal with respect to edge deletion. By construction, p_i corresponds to v_i, and we must pick a single edge e_{v_i} from p_{i-1} to p_i. Therefore, each v_i has a single edge in $\mathcal{H}$.

Similarly, the only way to get from q_0 to the target, f, is through the q vertices. Since $\mathcal{H}$ contains a single path from p_0 to f, and since we force the edge $(\{p_n\}, \{q_0\})$, there must be a single path from q_0 to f. The three vertices $q_{i,1}, q_{i,2}, q_{i,3}$ only have edges to q_{i+1} or f. Therefore, we must pass through exactly one q_i vertex for each $i = 1 \dots m$. By construction, the three q_i vertices correspond to the clause c_i and we must pick a single edge e_{c_i} from q_{i-1} to q_i. Therefore, each c_i has a single edge in $\mathcal{H}$. □

Lemma 5.5.5 *A hyperpath $\mathcal{H}$ in $C(\phi)$ corresponds to a variable assignment that satisfies ϕ.*

Proof. Lemma 5.5.4 says we pick a single edge e_{v_i} for each variable v_i. By construction, $C(\phi)$ contains two edges for each v_i. Exactly one of these edges corresponds to assigning v_i to true, and is connected to all the clauses where v_i appears in negated $\neg v_i$ form. The other of these edges corresponds to assigning v_i to false, and is connected to all the clauses where v_i appears in positive form.

If we pick the edge that corresponds to assigning v_i to true, then all the clauses in which $\neg v_i$ appears cannot be satisfied by our assignment to v_i. They must be satisfied by at least one of the other two literals in the clause. Similarly, if we assign v_i to false, it cannot satisfy clauses where v_i appears in positive form.

Each edge $e_{c_i} \in \mathcal{H}$ is the single edge we pick for a clause c_i (Lemma 5.5.4). This edge corresponds to deciding which of the three literals satisfies c_i. If there is some literal that satisfies a clause under a variable assignment, then there is some valid choice for e_{c_i}.

If there existed a valid hyperpath solution in $C(\phi)$, then we could assign the variables in ϕ the corresponding values to e_{v_i} for each v_i. Then, each clause c_i would be satisfied by the literal e_{c_i}. This would be a variable assignment that satisfies ϕ. □

Lemma 5.5.6 *A hyperpath $\mathcal{H}$ in $C(\phi)$ that corresponds to a variable assignment that satisfies ϕ exists **if and only if** there is a solution to ϕ.*

Proof. Lemma 5.5.5 shows that if we have a solution to $C(\phi)$, then we have a solution to ϕ. We now show that if there is a solution to ϕ, then there is a solution to $C(\phi)$.

Suppose there is a variable assignment satisfying ϕ that has no valid hyperpath solution in $C(\phi)$. Since variables are assigned a value, it still holds that we have exactly one edge $e_{v_i} \in \mathcal{H}$ for each variable $v_i \in \phi$, that corresponds to a true or false assignment of the variable. A variable assignment satisfying ϕ that has no valid hyperpath solution in $C(\phi)$ then means that for at least one clause c_i, there is no valid choice for e_{c_i} as the literal which satisfies the clause.

However, if the variable assignment satisfies ϕ, then there must be at least one literal per clause which we can use to satisfy the clause. For each clause, we can therefore pick the edge that corresponds to the literal we use to satisfy the clause to create a valid hyperpath solution. This contradicts our initial supposition. □

Theorem 5.5.7 *The FHEP in acyclic F-hypergraphs is NP-complete.*

Proof. Our construction $C(\phi)$ is an acyclic F-hypergraph (Lemma 5.5.2).

Lemma 5.5.6 implies a hyperpath $\mathcal{H}$ in $C(\phi)$ from p_0 to f that must use $(\{p_n\}, \{q_0\})$ exists if and only if a solution to 3-SAT instance ϕ exists. We can construct $C(\phi)$ for any 3-SAT instance ϕ in polynomial time, thus 3-SAT is polynomial time reducible to FHEP on an acyclic F-hypergraph.

The 3-SAT problem is NP-complete. Therefore, the FHEP in acyclic F-hypergraphs is NP-complete via our reduction. □

Theorem 5.5.7 and Theorem 5.5.1 together imply that the SDHP in acyclic F-hypergraphs is NP-hard. Note that the problem is already known to be NP-hard when cycles are permitted [4]. Thus, the SDHP is NP-hard in F-hypergraphs in general.

Note that this implies the problem is also hard for acyclic BF-hypergraphs since they include acyclic F-hypergraphs.

One may be surprised that B-hypergraphs and F-hypergraphs have different complexity results. It seems intuitive that they are symmetrical: the directions of each edge could be reflected and s and d could be swapped. However, there is one subtle asymmetry that breaks this construction.

Hyperpaths are minimal with respect to edge deletion. So, in a hyperpath from s to d, there is only a single edge e such that $d \in H(e)$. However, they may be more than one edge $s \in T(e')$. This asymmetry means that finding a hyperpath in a reversed F-hypergraph may not be valid. Further, this property is sufficient to make the problem tractable in B-hypergraphs. Future research may find that this property is powerful enough to solve other algorithmic problems on B-hypergraphs.

To summarize, we find that computing the (s, d)-hypernetwork is NP-Hard in all cases, except for acyclic B-hypergraphs where it is solved in linear time. We summarize those findings by completing Table 5.1.

			Edge-dom	Input-dom	Dom
Regular	Cyclic	B	P (linear)*	P (linear)†	P‡
		F	P*	P†	P‡
		BF	P*	P†	P‡
	Acyclic	B	P (linear)*	P†	P‡
		F	P*	P†	P‡
		BF	P*	P†	P‡
Forced Edge	Cyclic	B	NP-Hard [4]	?	?
		F	NP-Hard [4]	?	?
		BF	NP-Hard [4]	?	?
	Acyclic	B	P (linear) [4]	?	?
		F	NP-Hard [22]	?	?
		BF	NP-Hard [22]	?	?

Table 5.2: Complexity summary for finding a hyperpath, revised.

*: Find the s-hypernetwork, prune edges.

†: Find the s-hypernetwork. Repeat with a subset of the input until done.

‡: Find the s-hypernetwork. Repeat with a subset of the input until done. Prune edges.

This brings us to the next section, where we formulate our redundancy problem as a SAT problem in order to try to solve it efficiently.

5.6 A SAT formulation of the redundancy problem

Consider the metagraph $S = \langle X, E \rangle$ in Fig. 5.5 and let's say we want to find a dominant metapath from $B = \{x_1\}$ to $C = \{x_5\}$, while forcing the metapath to use edge e_3. We can write this problem as $M_{dom}(\{x_1\}, \{x_5\}) \wedge e_3$. This will be the example illustrating our SAT formulation throughout the entire section. Note that this formulation consists in reality of SAT integer programming, where we try to minimize the number of edges in the metapath. As such, it pertains more to integer linear optimization than traditional SAT. More generally, the problem can be written as follows; Given a metagraph G, find $M_{dom}(S, T) \wedge e_x$.

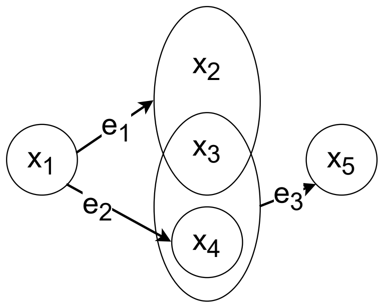


Figure 5.5: A simple example of a metagraph.

Representing variables and edges of the metagraph

First, we need a way to describe the variables and edges of the metagraph. For each of the edges and variables of the metagraph, we create a SAT variable.

Moreover, we introduce the notion of time steps. Time steps are chronologically ordered moments in time. We need this notion of time since we need to mimic the construction of a hyperpath in the graph. A formulation with no discrete time steps fails at building valid hyperpaths with a forced edge. Since we cannot use more than all the edges of the metagraph to build a metapath, the number of time steps is limited to the number of edges $|E|$.

Therefore, every SAT variable exists at every time step. Each variable x_i is instanced for each time step: $X_i = \{x_{i,0}, x_{i,1}, ..., x_{i,|E|-1}\}$. Since

Figure 5.5 contains three edges, each SAT variable v exists at three times: $\{v_{i,0}, v_{i,1}, v_{i_2}\}$.

Each variable is unlocked only once

When building a metapath, an edge cannot be used more than once. We represent this fact by adding constraints saying that every SAT variable can only be used at a single time step, or is not used at all.

More formally, for each set of variables $V = \{v_{i,0}, v_{i,1}, v_{i,2}\}$, only one of them must be true, or the variable must not be used: $(v_{i,0} \oplus v_{i,1} \oplus v_{i,2}) \vee (\neg v_{i,0} \wedge \neg v_{i,1} \wedge \neg v_{i,2})$. Algorithm 2 details the procedure to obtain this clause for each variable and edge in the metagraph.

Algorithm 2 Each variable is unlocked only once. The metagraph is mg.

```
procedure UnlockVariablesOnce($mg$)
    $timesteps \leftarrow$ len($mg.edges$)
    for $var$ in $mg.variables$ do
        $var\_clause\_parts \leftarrow$ []                ▷ Parts of clause for $var$
        for $t$ in range(0, $timesteps$) do
            $var\_once \leftarrow$ []                     ▷ $var$ used once
            for $t'$ in range(0, $timesteps$) do
                if $t == t'$ then
                    $var\_once$.append($var\_t'$)
                else
                    $var\_once$.append($\neg var\_t'$)
            $clause\_part \leftarrow$ " ∧ ".join($var\_once$)
            $var\_clause\_parts$.append("($clause\_part$)")

        $var\_unused \leftarrow$ []                       ▷ $var$ unused
        for $t$ in range(0, $timesteps$) do
            $var\_unused$.append($\neg var\_t$)
        $clause\_part \leftarrow$ " ∧ ".join($var\_unused$)
        $var\_clause\_parts$.append("($clause\_part$)")

        $var\_clause \leftarrow$ " ∨ ".join($var\_clause\_parts$)    ▷ Final clause for $var$
    constraints.append($var\_clause$)

    for $edge$ in $mg.edges$ do                   ▷ Do the same for edges
        ...
```

Since SAT does not support the XOR operator, we need to convert each of those clauses: $(x_{i,0} \wedge \neg x_{i,1} \wedge \neg x_{i,2}) \vee (\neg x_{i,0} \wedge x_{i,1} \wedge \neg x_{i,2}) \vee (\neg x_{i,0} \wedge \neg x_{i,1} \wedge x_{i,2}) \vee (\neg x_{i,0} \wedge \neg x_{i,1} \wedge x_{i,2}) \vee (\neg x_{i,0} \wedge \neg x_{i,1} \wedge \neg x_{i,2})$.

Unlocking variables and edges

Having determined each variable is used only once, we now tackle the problem of unlocking edges and variables. Indeed, we cannot allow the SAT solver to pick edges however it wants, we need to enforce the condition that using an edge requires the elements in its invertex to be reached by the source prior to the edge being used. Likewise, using a variable requires it to be reached by an edge prior to using this variable. This behavior can be found in the definition of the hyperpath, which we remind below.

Definition 5.6.1 (Hyperpath) *Let $\mathcal{H} = (V, E)$ be a directed hypergraph and let $s, t \in V$. A hyperpath from s to t in $\mathcal{H}$ is a minimal subhypergraph (where minimality is with respect to deletion of vertices and edges) $\mathcal{H}' \subseteq \mathcal{H}$. Further, the hyperedges comprising $\mathcal{H}'$ can be ordered in a sequence $\langle e_1, e_2, \dots, e_k \rangle$ such that for every edge $e_i \in \mathcal{H}'$ it is the case that $T(e_i) \subseteq \{s\} \cup H(e_1) \cup H(e_2) \cup \dots \cup H(e_{i-1})$ and $t \in H(e_k)$.*

An alternative way of understanding Definition 5.6.1 is to think about hyperedges *unlocking* vertices. We say that a hyperedge e_i in a hyperpath unlocks a variable $v \notin S$ if:

(i) $v \in H(e_i)$ and there is no hyperedge e_j that came before ($i > j$) such that $v \in H(e_j)$.
(ii) For each hyperpath edge, there must be some unlocked variable $v \neq S$ such that $v \in T$; or
(iii) v is utilized by some later hyperedge, i.e. there exists e_y ($i < y$) such that $v \in T(e_y)$; and
(iv) there is no e_x ($i < x < y$) such that $v \in H(e_x)$.

In other words, $\forall e_i, H(e_i) = \{x_1, \dots, x_k\}$, e_i *unlocks* $x \in H(e_i)$ iff.

$$\begin{aligned} \nexists e_j(j < i), v \in H(e_j) \text{ ; and} \\ (x \in T \text{ ; or} \\ \exists e_y(i < y), v \in T(e_y) \text{ ; and} \\ \nexists e_x(i < x < y), v \in H(e_x)) \end{aligned} \tag{5.1}$$

Those conditions can each be expressed in SAT as well. Algorithm 3 show how to express those constraints.

Clauses set to True

Some of the variables in the FHEP are already known at the start. This is the case of the source of the metapath S, the target of the metapath T and the forced edge e_x.

Since a solution to the FHEP requires elements in the source to be true, we can set them as True from the start: $\forall x_i \in S, x_{i,0} = 1$. Note that elements in the source are True at the first possible time step, since they are the start of the metapath.

The forced edge must also be True, but only at some point in time. More formally, $e_{x,0} \vee e_{x,1} \vee \dots \vee e_{x,|E|-1} = 1$

Same thing for elements in the target, they only need to be True at some point in time: $\forall x_i \in T, x_{i,0} \vee x_{i,1} \vee \dots \vee x_{i,|E|-1} = 1$

In the illustrating example, this would correspond respectively to $x_{1,0} = 1$, $e_{3,0} \vee e_{3,1} \vee e_{3,2} = 1$ and $x_{5,0} \vee x_{5,1} \vee x_{5,2} = 1$.

Algorithm 3 Generate clauses to unlock variables and edges. The metagraph is mg. $source$ and $target$ are respectively B and C.

```
1: procedure UnlockVariablesEdges($mg, source, target$)
2:     $timesteps \leftarrow$ len($mg.edges$)
3:     for $e_i$ in $mg.edges$ do
4:         for $u$ in range($0, timesteps$) do
5:             $cond\_i\_list \leftarrow$ []                                  ▷ Condition (i)
6:             for $var$ in $e_i.outvertex$ do
7:                 if $var \notin source$ then
8:                     for $e_j$ in $mg.edges$ do
9:                         if $e_i \neq e_j$ and $var \in e_j.outvertex$ then
10:                            $no\_edge\_before \leftarrow$ []
11:                            for $v$ in range($0, u$) do                ▷ $v < u$
12:                                $no\_edge\_before$.append($\neg e_j\_v$)
13:                            if $no\_edge\_before$ is empty then   ▷ $u == 0$
14:                                $no\_edge\_before \leftarrow$ ["True"]
15:                            $clause\_part \leftarrow$ " $\wedge$ ".join($no\_edge\_before$)
16:                            $cond\_i\_list$.append("($clause\_part$)")
17:             $cond\_i \leftarrow$ " $\vee$ ".join($cond\_i\_list$)
18:             if $cond\_i$ then
19:                 constraints.append($e_{i,u} \implies cond\_i$)
```

Clauses set to False

In the same way some elements can be set to True, other elements can be set to False. This is the case for unreachable variables and edges.

To obtain those, we simply run a linear time algorithm that explores the metagraph from the source. Each element that was not reached is by definition unreachable from the source, so we can set them to False. More formally, considering the set of unreachable elements $\mathcal{U}$, $\forall x_i \in \mathcal{U}, x_i = 0$.

Clause for input dominance

The metapath that we need to find in this version of the FHEP is input-dominant. Thus, we need to express this condition in SAT as well. Another way to think about input dominance besides the definition, is that each variable of the source must be *useful*. In other words, each variable of the source must have at least one of its outgoing edges using it at some point in the metapath. More formally, $\forall x \in S, \forall e \in E, x \in e.invertex, (e_{a,0} \vee e_{a,1} \vee ... \vee e_{a,|E|-1}) \vee (e_{b,0} \vee e_{b,1} \vee ... \vee e_{b,|E|-1}) \vee ... \vee (e_{n,0} \vee e_{n,1} \vee ... \vee e_{n,|E|-1})$.

Clause for edge dominance

To express edge dominance, we simply minimize the number of edges used in the metapath.

$$min(\sum_{i=1}^{n} e_i) \tag{5.2}$$

```

      **for** $var$ in $e_i.outvertex$ **do**
        **if** $var \notin source$ **then**
          **if** $var \in target$ **then**                    ▷ Condition (ii)
            *pass*
          **else**
            $cond\_iii\_list \leftarrow []$
            **for** $e_j$ in $mg.edges$ **do**
              **if** $e_i \neq e_j$ and $var \in e_j.outvertex$ **then**
                $cond\_iii\_ej\_list \leftarrow []$    ▷ Condition (iii)
                **for** $v$ in range($u+1, timesteps$) **do**    ▷ $v > u$
                  $cond\_iii\_ej\_list$.append($e_{j,v}$)
                $cond\_iii\_ej \leftarrow$ " $\vee$ ".join($cond\_iii\_ej\_list$)
                **if** $cond\_iii\_ej$ **then**
                  $cond\_iii\_list$.append("($cond\_iii\_ej$)")

                **for** $v$ in range($u+1, timesteps$) **do**    ▷ Condition (iv)
                  $cond\_iv\_list \leftarrow []$
                  **for** $e_k$ in $mg.edges$ **do**
                    **if** $e_i \neq e_k$ and $e_j \neq e_k$ and $var \in e_k.outvertex$ **then**
                      $cond\_iv\_ek\_list \leftarrow []$
                      **for** $v$ in range($u$+1, $timesteps$) **do**    ▷ $u < w < v$
                        $cond\_iv\_ek\_list$.append($e_{k,w}$)
                      $cond\_iv\_ek \leftarrow$ " $\wedge$ ".join($cond\_iv\_ek\_list$)
                      **if** $cond\_iv\_ek$ **then**
                        $cond\_iv\_list$.append("($cond\_iv\_ek$)")
                  $cond\_iv \leftarrow$ " $\wedge$ ".join($cond\_iv\_list$)
                  **if** $cond\_iv \leftarrow$ **then**
                    $cond\_iv$ += " $\wedge e_{j,v}$"
            **if** $cond\_iii\_list$ is empty **then**
              *continue*
            $cond\_iii \leftarrow$ " $\vee$ ".join($cond\_iii\_list$)

            **if** $cond\_iv$ **then**
              constraints.append($e_{i,u} \implies (cond\_iii \wedge cond\_iv)$)
            **else**
              constraints.append($e_{i,u} \implies cond\_iii$)
```

Final steps

After generating all those clauses, we need to convert them all to their Conjunctive Normal Form (CNF). This is because a SAT solver can only work with this kind of clauses.

We coded this SAT formulation of the FHEP. Unfortunately for us, it only worked on small instances of the problem. Larger metagraphs made the generation of the clauses too long to be useful for policy analysis. Namely, the conversion of formulas to their CNF form is the major hurdle performance-wise. Having demonstrated the FHEP is NP-Hard, and that the SAT formulation is not very efficient, we now present an algorithm more efficient than both the SAT formulation and the existing solution in the literature.

5.7 Towards a more efficient approach using Pascal's triangle

5.7.1 The case of Hasse diagrams

After considering initial leads, we tried to find a structure that could best exploit the definition of dominant metapaths. By definition, once you find a dominant metapath, all supersets of the edges composing the dominant metapath will compose a redundant metapath, since the added edges are not necessary for connectivity. This means that checking all the supersets for dominance can be eluded. Hasse diagrams, used to represent partially ordered sets seemed indicated. Fig. 5.6 represents the power set of a 4-element set ordered by inclusion. Bit strings in nodes represent a combination of elements of the set. For example, the 0000 bit string represents the empty set, while the 1000 bit string represents the set containing one of the elements of the set.

In the context of our problem, elements in a set represent edges of the metagraph. It follows that, if a dominant metapath is found in the diagram, all supersets are redundant metapaths. For example, if the metapath consisting of an edge represented by the bit string 1000 is dominant, there is no need to consider any superset of 1000, such as 1100, 1011, or 1111. Even though the complexity of the construction of the power set is exponential in the number of edges, we hope that finding dominant metapaths will prune the diagram enough so that the computation time remains reasonable even for large graphs.

To construct a Hasse diagram, we start at the line of sets containing only one edge, and construct levels on top iteratively, one by one. We iterate over two sets of the level at a time, and combine them (with an union) to form a set of the superior level. Every time a new set is created, if we did not check it already before, we check if it is a dominant metapath. If it is a dominant metapath, we add it to the list of found metapaths, otherwise we add the newly created set to the list of sets considered for combination in the next iteration/level of the Hasse diagram. Note that there is no need to model the diagram as nodes and edges, we only need the sets of a level to be able to find metapaths and build additional levels. This method is not very efficient, which leads us to our proposed solutions.

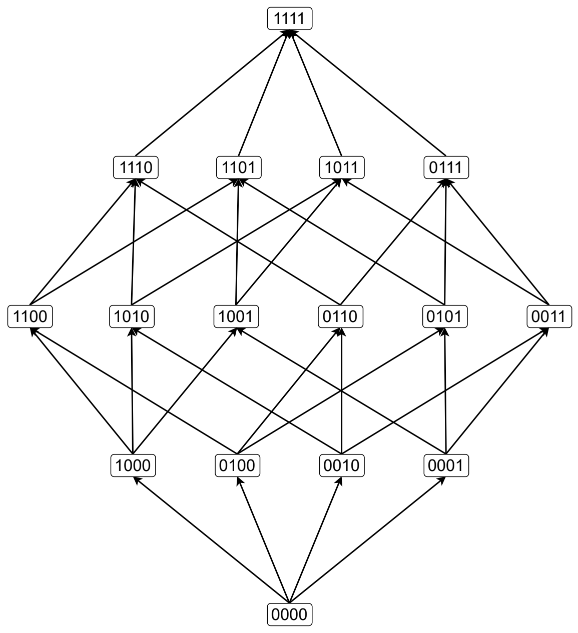


Figure 5.6: Hasse diagram representing the power set of a 4-element set ordered by inclusion.

5.7.2 Proposed solution 1: prefix trees

Even though Hasse diagrams proved useful conceptually, there are some optimizations we can make. It is wasteful to consider two elements at a time when constructing a level of the Hasse diagram, since some combinations yield the same resulting set, and some other combinations give sets belonging to a level not directly above the one we are iterating on. Conceptually, the resulting graphical representation is more akin to a prefix tree, represented in Fig 5.7.

Here, we build additional levels by considering each set in the current level, and then considering which edge we can add to the set. This is indicated by the zeroes in the bit strings. For example, the set 1000 has three zeroes and thus, we form the sets 1100, 1010 and 1001 by adding an edge in those positions. Note that the next set in the level, 0100 also builds 1100 which is a problem since we want to avoid repetitions. To this end, we also add a bit mask when building new sets, which masks bits up to the last 1 in the bit string, since those sets were already created by the previous constructions. For example, the mask for bit string 0110 will be 1100, such that we only build 0111 and not 1110 which is already built from 1100.

Like with the Hasse diagram method, we check sets for dominance, and add them to the list of found dominant metapaths if they are, or add the sets to the next level otherwise.

5.7.3 Proposed solution 2: Pascal's triangle

In this alternative solution, we propose to use properties of Pascal's triangle to achieve better results. A problem with the prefix tree method is that the construction is not balanced. It is easy to see on Fig. 5.7 that some sets prune more nodes than others if they are dominant. For example, if the set 1000 is dominant, we can see that seven sets in superior levels are not considered anymore. However, if the set 0010 is dominant,

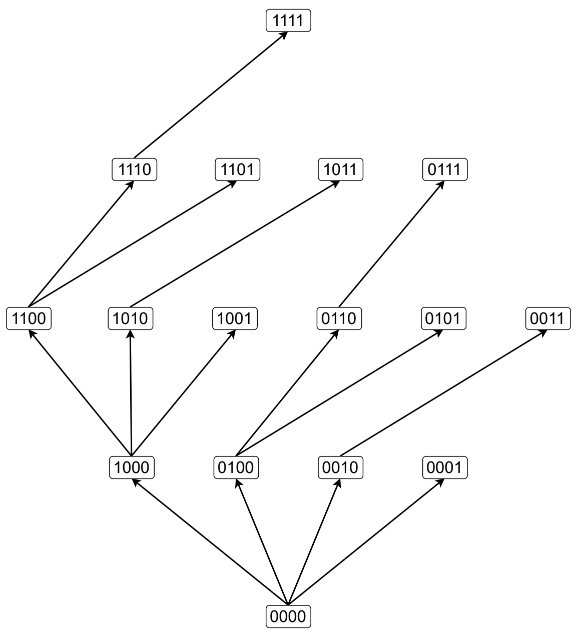


Figure 5.7: Prefix tree representing the power set of a 4-element set ordered by inclusion.

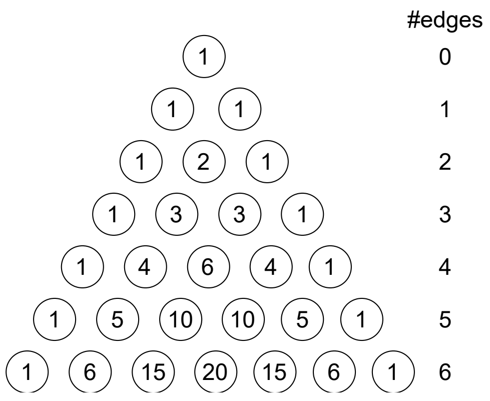


Figure 5.8: Pascal's triangle.

only one set is not considered anymore. This is a product of the way we build the prefix tree, and not a product of the sets since sets in a level can only be linked to some, not all, of their supersets in the level directly above, e.g. 0010 is not linked to either 1010 or 0110.

We might also consider sets which we do not need to consider. For example, if set 0110 is dominant, we know set 1110 is not dominant, but is still built by set 1100 and checked for dominance since 1100 comes first in the list. Both those issues are problematic since we do not get the full effect of finding a dominant metapath for some sets.

In this second solution, we solve this issue. Each level of Pascal's triangle (Fig. 5.8) corresponds to the number of sets of a specific size for a specific number of elements. To give an example, the level of Pascal's triangle corresponding to four edges is $1, 4, 6, 4, 1$, which corresponds to the number of sets in each level of the prefix tree in Fig. 5.7.

We propose to build the sets in levels of Pascal's triangle, starting with one edge, and adding an edge at each iteration. We thus build power sets for an increasing number of edges. To build the next level in Pascal's triangle, we perform an operation on sets of the previous level. In the same way an element not on the edge of the triangle is created by adding its two parents, we create a set by combining the parents of the set.

More specifically, we add the edge of this level of Pascal's triangle to sets of the left member, and then add the sets of the right member: $new\_set = (new\_edge * left\_member) + right\_member$.

For example, imagine we have all the sets of the level consisting of four edges, and want to build the fifth level by adding an edge. Let us define the four initial edges as a, b, c, d and the new edge we want to add as e. We need to build the sets corresponding to the level with five edges: $1, 5, 10, 10, 5, 1$. Sets at the edge of the triangle (1 on the left and 1 on the right) correspond to respectively the empty set and the complete set. Sets inside the triangle can be built from the previous level. For example, the 10 on the left is built by combining the 4 and 6 of the level with four edges. The 4 corresponds to sets $\{a\}, \{b\}, \{c\}, \{d\}$, and the 6 corresponds to sets $\{a, b\}, \{a, c\}, \{a, d\}, \{b, c\}, \{b, d\}, \{c, d\}$. To obtain the 10 in level 5, we combine the new edge e with sets in 4. We obtain $\{a, e\}, \{b, e\}, \{c, e\}, \{d, e\}$. Like with the other methods, those sets are checked for dominance and only added to the level if they are not. Then, we add the sets in 6 to obtain the sets of this particular element of Pascal's triangle.

In addition to this, we also optimize inside a level of Pascal's triangle. Like with the prefix tree method, if a dominant metapath is found in the level, supersets do not need to be considered for dominance, so we keep a memory of the bit strings representing dominant metapaths. If a dominant metapath is found in the level, we compare newly the intersection of created sets with found dominant metapaths. If the intersection is equal to a found dominant metapath, the newly created set is a superset, does not need to be checked for dominance and can be removed from the level.

By doing the combination this way, we can make sure to maximize the effect of finding a dominant metapath. In the previous example, if the set $\{b, e\}$ was found dominant, it would simply not be added to the list of sets, and thus will never be used in other combinations.

5.8 Performance analysis

This section is dedicated to the performance evaluation of the different algorithms we have reviewed so far. Namely, we measure the speed of the algorithm given in Ranathunga *et al.* [28], the algorithm enumerating all metapaths, the algorithm using a SAT instance, and the algorithm using Pascal's triangle. The enumeration algorithm is evaluated as to have a baseline when comparing the other algorithms. We modify the algorithm given in Ranathunga *et al.* as well as the enumeration algorithm as to not detect conflicts, only redundancies. This will put those algorithms on a more equal footing as the algorithms using SAT and Pascal's triangle do not detect conflicts.

5.8.1 Methodology

We perform policy analysis to detect redundancies in YAWL policies. In each case, we start with a metagraph and the same source and target, respectively the start and the end of the workflow, as reference for the elimination of redundancies. Methods to detect redundancies vary. Ranathunga *et al.* [28] algorithm and the enumeration algorithm are given the metagraph. The algorithm using Pascal's triangle is given the metagraph, the source and the target. The SAT formulation is given all of the above and a specific edge to force.

To obtain general and representative results, we generate random workflows. Like in Chapter 4, this allows us to get an idea of the efficiency of the algorithms. Thus, the generated workflows should not display specific features that would give an edge to one algorithm or the other.

We perform the analysis of those policies by varying the number of elements in the workflow. In particular, we modulate the number of elements we generate, by giving them values 5, 10 or 15. Those values were chosen after empirical evaluation, where it was clear all algorithms had shown their limits. Therefore, there is no use to adding more values for the number of elements. In opposition to Chapter 4, we do not vary the policy size, as it was shown in this chapter to have little impact on execution time. The policy size here is thus set to 0, where we have one conditional proposition on each edge. We also introduce no translation error rate since there is no translation involved, only the analysis of the YAWL specification.

Since we vary the number of elements we generate, this also varies the number of edges in each specification. We use the same ratio as in Chapter 4, 1.5 times more edges than the number of elements. Overall, we generate thirty random YAWL specifications for each size of the workflow, creating 90 different workflow specifications in total (3 generating set sizes, 30 repetitions). We ran our measurements on a commodity desktop, equipped with an Intel Core CPU 3.5-GHz, 16GB of RAM and running Ubuntu 20.04.

Once those specifications are generated, each one can be turned into its conditional metagraph representation, and we can run each algorithm on them. The only exception is the SAT formulation. In this case, we first need to transform the workflow specification into a SAT instance, and then we can use a solver to find a solution to this instance.

5.8.2 Evaluation

SAT generation times

First, we give the generation times for instances of the SAT formulation. The distribution of generation times can be found in Figure 5.9. As the reader can notice, either generation times approach zero seconds, or 1800 seconds (30 minutes). Empirically, we noticed that running the generation for longer periods of time than those 30 minutes did not allow for the generation of the SAT instance to be finalized. Therefore, we put in place a timeout that would end the generation of the SAT instance should it go over 30 minutes.

Concerning 5-element workflow SAT instances, 5 of them out of 30 were unable to complete. In opposition, all 30 of the 10-element workflow SAT instances could not be generated. As a result, we only kept generation times of SAT instances for workflows with 5 or 10 elements, resulting in 60 generated SAT instances. As far as we can tell, the fact that some generations were not able to be completed is due to their conversion to Conjunctive Normal Form. Perhaps the conversion has a large middle, with many terms, and struggles to complete the CNF. A potential way to fix this issue would be to apply De Morgan's laws manually.

Figure 5.10 gives a clearer view of the distribution times of Figure 5.9, by excluding all the generations that timed out. Note that for SAT instances that were able to finish, generation times are very low.

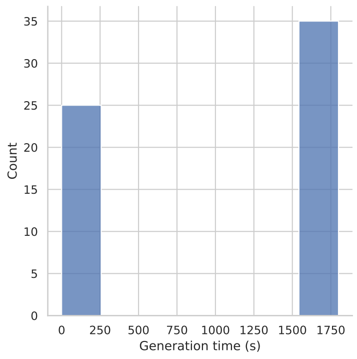


Figure 5.9: Distribution of generation times for SAT instances.

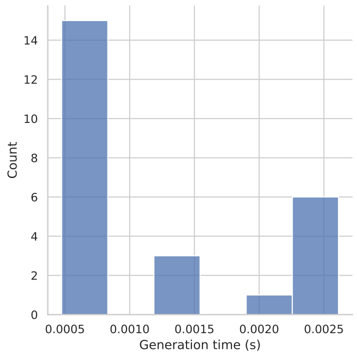


Figure 5.10: Distribution of generation times for SAT instances, minus the ones exceeding the time limit.

Redundancy detection: a comparison

Figure 5.11 depicts the execution times of the measured policy analysis algorithms, according to the number of edges in the workflow specification. Like with the generation times of the SAT instance, we tested empirically that after one hour, there was no use in continuing to run the algorithms. Therefore, we also set a timeout of one hour, which is why the algorithms level off at the 3600 second mark. As we can see, the Ranathunga *et al.* algorithm is in line with the enumeration algorithm in terms of speed. Recall that the Ranathunga *et al.* algorithm does not detect most redundancies. In opposition, the enumeration and Pascal triangle algorithms detect all redundancies. We can see that if the specification contains more than 10 edges, we cannot compute redundancies with the Ranathunga *et al.* algorithm or the enumeration algorithm anymore as they level off. In the same way, the algorithm using Pascal's triangle levels off after 28 edges, which grants us access to the computation of redundancies on larger metagraphs, as well as better execution times for metagraphs of the same size when compared to the enumeration algorithm.

Concerning the SAT formulation, we were able to measure its performance on instances that we were able to generate. For the instances we were not able to generate, we simply set their execution time to one hour, the value of the timeout. We can observe the execution time for SAT levels off faster than the enumeration algorithm. Note that for instances that we were able to generate, the time to solve the SAT instance is very low. If we could somehow find a way to find a solution to the problem that prevents us from generating SAT instances for larger metagraphs, we could potentially find that the execution time to solve SAT instances stays low. As of now, the fact that SAT levels off after only 8 edges is simply

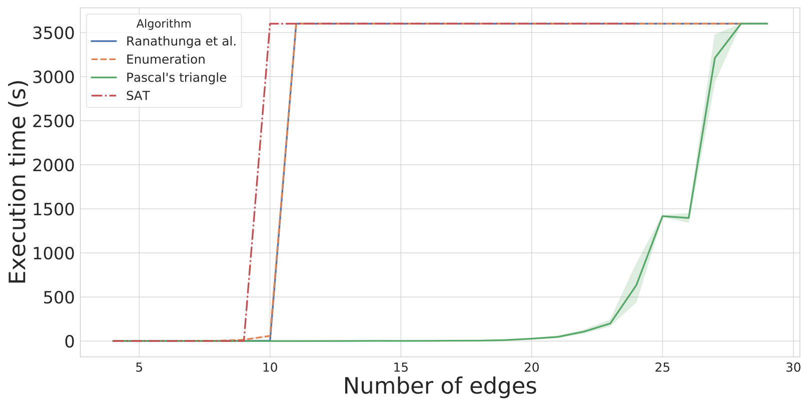


Figure 5.11: Execution times of different policy analysis algorithms. In particular, we compare Ranathunga *et al.* [28], the algorithm enumerating all metapaths, the SAT algorithm and the algorithm using Pascal's triangle.

due to the fact that instances with more edges were not generated, and not due to the fact that the actual resolution of the instance takes a lot of time.

Overall, we find that the algorithm using Pascal's triangle is indeed the most efficient out of the tested algorithms, since it detects redundancies, has a larger capacity in terms of metagraph size, and better execution times than the enumeration and SAT algorithms. The limit of this algorithm is that the number of edges on paths from the source to the target should not exceed 28 edges. This should be enough when dealing with security policies, as they will rarely have more than 28 relations between sets of elements to evaluate for an access control decision, but could be problematic for very large workflows. In the case of this YAWL foundation case study[49] for example, the workflow contains around 50 edges from start to finish. The complete data, code to generate the measures and results, as well as some guidance are publicly available[4] .

[49]: Business Process Management (BPM) Group (2010), *YAWL4Film*

4: See `https://github.com/loicmiller/policy-analysis`.

5.9 Discussion

In this chapter, we addressed the problem of finding redundancies in a policy specification. Specifically, we investigated the currently existing solution and showed it was inadequate because it was manual, partial, and relied on computing the transitive closure of the adjacency matrix which is O($(n^3)^m$), with n the number of metagraph variables and m the length of the longest path.

From this result, and since we could not come up with an algorithm that would find redundancies in a reasonable time, we extrapolated that maybe our problem was NP-hard. We showed this was indeed the case, by studying the equivalent problem of finding (s, d)-hypernetworks in the general case and in acyclic F-hypergraphs.

In the process of studying the complexity of those problems, we merge the field of metagraphs and hypergraphs, bringing metagraph concepts like edge dominance and input dominance into the realm of hypergraphs. We hope that by bridging the gap between those constructs, we can further both fields of research as elements of metagraphs can be used to solve hypergraph problems and vice-versa. For example, since finding (s, d)-hypernetworks is a core problem, NP-hard in the general case and polynomial only in acyclic B-hypergraphs, there might be some concrete application of this complexity boundary.

The only case in which finding redundancies is feasible in polynomial time is in the case of acyclic B-hypergraphs. However, this is only a very specific type of graphs and will not be able to model a large portion of policies. Indeed, the B-hypergraph only contains B-edges, which represent many-to-one relationships between elements. A B-hypergraph is not capable of modeling one-to-many relationships in the way F-hypergraphs can, or many-to-many relationships, like BF-hypergraphs can. As such, many of the common relationships that can be found in policies cannot be accurately represented in B-hypergraphs. On the contrary, we argue most policies probably need to be represented using a BF-hypergraph. Since finding redundancies is only feasible in polynomial time when the B-hypergraph is acyclic, this means the policy we want to analyze cannot contain any cycles. This is limiting, as a policy analysis method using metagraphs will in theory only be efficient in acyclic B-hypergraphs, and in general a policy cannot be accurately represented using only a B-hypergraph.

To find an efficient solution to our redundancy finding problem, we formulated it as a SAT problem. By doing this, we hoped that a SAT solver could find a solution efficiently in most cases, and to be honest I feel this is still probably the case. The problem lies is the number of clauses we generate to formulate the problem as SAT, which is exponential. In accordance with this intuition, cases where we could generate the clauses were solved very efficiently, but we do not know the efficiency of cases where we could not generate the clauses. One thing we could have done differently was to formulate our problem as SAT without going through the metagraph representation. We hoped that by going through this representation first, we could come up with a simpler and more effective way of finding redundancies. Another limitation is the fact that potentially, multiple SAT formulations exist for our problem. Maybe our current formulation is one of the less effective ones, and maybe another way of tackling the problem would prove more efficient.

Finally, we propose an algorithm as an alternative to the existing one of the literature and our SAT formulation. This algorithm uses Pascal's triangle, and takes the exhaustive approach. Since the algorithm works on the subset of the metagraph that is relevant to the access request being tested (e.g. attributes of the user, firewalls between the user and the resource), this helps us reduce greatly the number of tested metapaths.

By measuring performance of the algorithm given in Ranathunga *et al.* [28], the algorithm enumerating all metapaths, the algorithm using a SAT instance, and the algorithm using Pascal's triangle, we find that the algorithm using Pascal's triangle is the most efficient out of the tested algorithms. Indeed, it detects redundancies, has a larger capacity in terms of metagraph size, and better execution times than the enumeration and SAT algorithms.

A limitation coming with the exhaustive approach when the problem to solve is NP-hard, is that the algorithm will have non-polynomial complexity. This is the case of our solution, which has exponential complexity. We argue however that this is an efficient solution for the exhaustive approach, as we do not need to calculate the $A*$ matrix, and we test the least amount of edge combinations.

When testing our algorithm, we find that it can handle access control policies well, but can struggle with workflows. Indeed, one limit of the algorithm is that the number of edges on paths from the source to the target should not exceed 28 edges. This should be enough when dealing with security policies, as they will rarely have more than 28 relations between sets of elements to evaluate for an access control decision, but could be problematic for very large workflows. In the case of this YAWL foundation case study[49] for example, the workflow contains around 50 edges from start to finish. Again, it is not the size of the policy that matters when using our algorithm, but the number of edges relevant to the access being tested.

[49]: Business Process Management (BPM) Group (2010), *YAWL4Film*

5.10 Conclusion

In this chapter, we addressed the problem of finding redundancies in a policy. We showed how the currently existing solution was inadequate, and proved the problem of finding redundancies in a policy is actually NP-Hard in the general case. We fill the gap in the literature by studying the equivalent problem of finding (s, d)-hypernetworks in the general case and in acyclic F-hypergraphs. Knowing this problem is NP-hard, we tried to formulate our redundancy detection problem as a SAT problem, with little success. In spite of this result, we proposed an alternative method using Pascal's triangle which still has exponential complexity, but has the merit of being thorough and exhaustive. We then confirm the efficiency of this algorithm by measuring and comparing the performance of the algorithms described in this chapter.

A part of this work, the proof of hardness on acyclic F-hypergraphs, is currently in submission [22]. We also developed publicly available tools, namely the SAT formulation, and a policy analysis framework containing the other redundancy detection methods introduced in this chapter (the algorithm of Ranathunga *et al.*, the enumeration algorithm, and the algorithm using Pascal's triangle), code to generate workflows, code to perform the measurements and other related functionalities.

Overall, we argue that although metagraphs are fine when analyzing a security policy, they might not be the best way to find redundancies in a workflow. Possible future works include finding a different SAT formulation of the problem, which does not need an exponential amount of clauses. In particular, Algorithm 3 needs to be simpler and generate less clauses, and Algorithm 2 generates clauses that expand exponentially when they are converted to their CNF. Other possible future works include using metagraphs/hypergraphs for other policy analysis problems. Namely, one possible aspect which has not yet been explored is the detection of policy incompleteness using metagraphs. Policy incompleteness has been studied with decision diagrams, but there may be a way to find a more efficient method using metagraphs.

Conclusion and Research Directions 6

6.1 Summary of our works

In Chapter 1, we introduced workflow and concepts related to their security. We explained that most, if not all organizations use workflows in their daily operations, and that workflows can involve multiple organizations. Those multi-party workflows are complex to implement when it comes to communication, management and security.

We showed how data exposures are increasing in numbers and severity, and constitute a critical vulnerability. We explained that those exposures are aggravated because of the shift in the way applications are currently developed and deployed. Indeed, more and more organizations take a modular rather than monolithic approach, and drop on-premise deployments in favor of the cloud and microservices. This in turn has led to a change in the way security must be envisioned, with models like zero-trust being more adequate than the typical *"protect the border"* approach. We briefly described typical countermeasures, that come in the form of authentication and authorization.

Considering the current situation, with major companies storing their data unencrypted in the cloud, we took interest to the security of workflows in such an environment. Our goal was to enable secure multi-party workflows and mitigate the risk of data exposures. In accordance with this goal, we structured this document around three axes.

An infrastructure to prevent exposures

To mitigate risks of exposures, we proposed an infrastructure using microservices in Chapter 3. In particular, we provide ways to secure data at rest and in transport, and a way for the workflow manager to enforce a policy.

After going through the different mechanisms and explaining how each of them tends to our goal, we deployed a Proof of Concept on the Google Cloud Platform and verified our policy implementation is actually enforced via captures made at central points in the infrastructure. Measuring the cost of the added authorization, we find that pods with Open Policy Agent have a substantial increase in startup time of almost two seconds on average (32.72%). Measuring request time according to the size of the policy, we find that policy size accounts for 65% of the variance in intra-region communications whereas it accounts only for 7% of the variance in inter-region communications. This work has led to two publications [19, 20] and the development of a publicly available Proof of Concept.

Our infrastructure assumes its policy specification to match its implementation. In reality, this can be erroneous, and might lead to potential attacks exploiting a faulty policy.

Verifying policies

To drop this assumption, we dealt in Chapter 4 with the verification of policies. In this chapter, we detailed a way to verify that the policies deployed in our infrastructure actually match their initial specification. To this end, we introduced metagraphs, and gave arguments as to why it was an appropriate way to model policies, workflow policies in particular. We showed we can fully express YAWL with metagraphs, and how each task, condition and operator can be converted into a metagraph representation, supporting the richness of the YAWL language. We used YAWL and Rego as our go-to languages respectively for the policy specification and implementation. We then introduced a method to enable policy verification with the help of metagraphs. We evaluated our verification method, and found it to run in a very reasonable time, even with relatively large policies.

This work has led to two publications [20, 21], the development of a Python 3 version of MGToolkit and the development of a policy verification framework. Those tools are publicly available.

Our verification method also had assumptions. Namely, we assume the policy specification to be free of errors and redundancies. This is far from always being the case. Our verification method can only conclude that the implementation of a policy matches its specification, but does not allow for the detection of errors and redundancies in the specification only.

Analyzing policies

This is why in Chapter 5, we addressed the problem of finding redundancies in a policy. We showed that the current solution is inadequate, and proved finding redundancies in a metagraph is NP-hard. Based on this finding, we formulated our problem as a SAT problem to have a more efficient way of finding redundancies, but found this method to be inefficient. As an alternative, we proposed an exhaustive algorithm using Pascal's triangle to find redundancies, which is more efficient than the previous solution. To confirm this finding, we conducted a performance analysis to compare those different methods of finding redundancies. As expected, our method is faster and more scalable than the others. A part of this work, the proof of hardness on acyclic F-hypergraphs, is currently in submission [22]. We also developed publicly available tools, namely the SAT formulation, and a policy analysis framework containing the other redundancy detection methods introduced in this chapter (the algorithm of Ranathunga *et al.*, the enumeration algorithm, and the algorithm using Pascal's triangle), code to generate workflows, code to perform the measurements and other related functionalities.

6.2 Takeaways from this thesis

The main takeaways of this thesis are as follows:

- **The microservice architecture can be used to construct a leak-free multi-party workflow.**
  We show how to use the microservice architecture to secure workflows. We propose a secure infrastructure and deploy it on Google Cloud Platform in a publicly available Proof of Concept. With this infrastructure, we hope to mitigate data exposures by dealing with cryptographic failures, and providing a way to enforce a policy. Our infrastructure does not however protect against all attacks, and has some assumptions. We assume in particular that the policy specification is correct, and matches its implementation.

- **Metagraphs are a useful structure to model workflow policies, and can be used to perform policy verification.**
  To the best of our knowledge, this is the first time someone has shown how to verify policies using metagraphs. Metagraphs are a relevant and efficient way to model, manage and verify deployed policies, workflow policies in particular. We showed we can fully express YAWL with metagraphs, therefore supporting the richness of the YAWL language in metagraphs. By verifying the specification matches the implementation, we hope to reduce the number of erroneous implementations and thus mitigate data exposures due to broken access control.

- **Metagraphs are a useful structure to analyze policies, in particular to identify and remove redundancies.**
  The current way to detect redundancies with metagraphs has some issues. We showed analyzing a policy to find redundancies is NP-hard in the general case, and presented an algorithm using Pascal's triangle. We argue this is an efficient solution to this problem, and confirm this method is better than other ways of finding redundancies by conducting a performance evaluation. We hope that by providing ways to identify redundancies, we can further reduce the risk of broken access control in deployed policies.

All the tools, as well as any data, code to generate measures, results and guidance, are publicly available in their associated repository. Links to those can be found in Table 1.2. Data for the measures of Chapter 4 takes too much space for a git repository and can be found here instead.

6.3 General discussion

We discussed each of our contributions in their respective chapters. In Chapter 3's discussion, we looked back on our attacker model, and show how each attacker is countered by mechanisms of our infrastructure. We detailed other benefits of our infrastructure, such as its modularity or reduced complexity compared to monolithic solutions. To the best of our knowledge, this is the first time someone showed how to use the microservice architecture to secure workflows. Some threats still remain, as someone using an unsecure outdated version of any service within our infrastructure will be vulnerable to exploits possible on this version of the service. Since the service inside our infrastructure can be literally anything, there is little we can do to prevent those attacks. One of our assumptions is that policy specification is correct, and its implementation corresponds.

We dealt with this assumption in Chapter 4 by building a policy verification method. We evaluated the performance of our solution, and come to the conclusion that using metagraph comparison to verify deployed policies match their specification can be performed in a very reasonable time, even for a large number of rules. The main point to discuss here is the nonexistence of any representative policy dataset existing online. To reiterate, most policies we found were either contained in articles, or in the wild in public GitHub repositories. The majority of papers dealing with policies do not disclose them, as they are either private or currently in use. There isn't any publicly available policy dataset, which contain policies representative enough to conduct analysis on them. The assumption here is that the policy specification is correct.

This is dealt with in Chapter 5, where we detail ways to find redundancies in a policy. A limitation coming with the exhaustive approach when the problem to solve is NP-hard, is that the algorithm will have non-polynomial complexity. We argue however that our solution is efficient for the exhaustive approach, as we do not need to calculate the $A*$ matrix, and we test the least amount of edge combinations. In our search for eliminating redundancies, we also brought metagraph concepts like edge dominance and input dominance into the realm of hypergraphs. We hope that by bridging the gap between those constructs, we can further both fields of research as elements of metagraphs can be used to solve hypergraph problems and the other way around as well. To conclude on the algorithm using Pascal's triangle, we tested our algorithm and found that it could handle access control policies well, but could struggle with workflows. Restating one of our findings, one limit of the algorithm is that the number of edges on paths from the source to the target should not exceed 28 edges. This should be enough when dealing with security policies, as they will rarely have more than 28 relations between sets of elements to evaluate for an access control decision, but could be problematic for very large workflows.

When considering the works realized in this document, we feel that there is still a lot of progress to be made, but that it is nonetheless a step in the right direction. When combined, we have a secure infrastructure that mitigates the risk of data exposures, with a way to enforce a policy implementation that can be verified, based on a policy specification that can be analyzed. We feel strongly about the use of metagraphs to represent policies, workflows in particular, and have argued why in this document. Metagraphs are a great way to model policies, and the more one can achieve with them, the more useful they become.

In our works, we also took a very inclusive definition of workflows. Indeed, a lot of workflow patterns can be expressed using graphs, conditional metagraphs in particular, so it seemed fitting. However, one aspect that would merit further investigation is the influence of workflow patterns on our different contributions. A conditional metagraph should be able to model those patterns, as they are general-purpose, but there might be some important implications in the policy verification and analysis processes which we could leverage to better verify and analyze workflows.

Overall, we feel like the stronger part of our works are the ones presented in Chapter 4 and Chapter 5. Looking back at the results presented in this thesis, we should perhaps have dedicated a smaller part to the secure infrastructure. In opposition to Chapter 4 and Chapter 5, Chapter 3 is definitely more of an engineering problem than a research problem. A lot of private companies use microservices, but there is little work on their security guarantees, and the body of research around microservices is not consequent even though their are ubiquitous. Those companies have little incentive to publish their work, and it can be hard for a research unit to compete with teams of engineers at major businesses.

6.4 Perspectives and future works

Future works for each contribution were addressed in their corresponding chapter. For Chapter 3, possible future works include studying how changes in the workflow impact the security of our infrastructure, or trying to remove some of the trust requirements. A possible lead to lowering those requirements would be to use Trusted Execution Environments (TEE), which are a secure area inside a processor.

Concerning Chapter 4, possible future works include broadening the types of policies our verifier can handle. To extend the domain of policies we are able to verify, a possible lead is to investigate pattern matching solutions in order to provide a more general way of interpreting identifiers. Another important lead is the construction of a representative publicly available policy dataset.

As for Chapter 5, possible future works include finding a different SAT formulation of the problem, which does not need an exponential amount of clauses. One thing we need to try is a manual application of De Morgan's laws, which should reduce the time to convert clauses to their Conjunctive Normal Form. In addition, the complexity of our current solution using Pascal's triangle is unfortunately still exponential. To mitigate this and improve our algorithm, we could use a version of Tarjan's algorithm adapted to metagraphs in order to consider only edges on a path from the source to the destination, since other edges will not be part of a dominant metapath. In a metagraph, we define a strongly connected component (SCC) as a subset of variables where every variable in the subset can reach other variables in the subset using a simple path. To easily find edges on a path from the source to the destination, we could add a virtual edge from the destination to the source, then run Tarjan's algorithm to find SCCs. The SCC containing the source and destination will contain the edges to be considered when running the algorithm Pascal's triangle. Since the SCC will contain less edges than the full metagraph, we can probably gain a little performance this way. Alternatively, one could also perform a DFS from the source to find nodes reachable from the source, and a DFS from the destination in the transpose metagraph to find nodes reachable from the destination. Then, the intersection of the nodes yields the nodes on a path from the source to the destination.

Other possible future works include using metagraphs/hypergraphs for other policy analysis problems. Many properties have not yet been explored with those constructs, like the separation of duties or the principle of least privilege. Another one of those aspects is the detection of policy incompleteness, where one tries to identify policy requests that have no explicit decision.

Overall, our short term goals include finding other ways to model the policy analysis problem as SAT/ILP. We only proved this problem was NP-hard, but it might belong to a higher complexity class in the hierarchy, so there might be something to do here. Midterm goals include researching how several workflow patterns impact our contributions. Cancellation in particular seems an interesting pattern to investigate. We also plan to continue exploring complexity results occurring in metagraphs and hypergraphs. For example, we think the related problem of finding s-hypernetworks can be done quicker for certain types of hypergraphs, and the complexity of this procedure has yet to be determined for other types of hypergraphs. Finally, goals in the longer run include the constitution of a representative publicly available policy dataset. We are not alone in thinking this is a crucial missing part of the current body of research. We also plan to investigate more policy-related issues, and try to tackle them using metagraphs. This includes policy properties like verifying separation of duty, or finding incompleteness in policies. One aspect that would merit further investigation as well is the influence of workflow patterns, like cancellation, on our different contributions. In the future, we hope the growing body of work around policies, workflows and metagraphs will be enough of an incentive for policy administrators to use them in practice.

Appendix

A Complementary Materials for Chapter 3

A.1 Listings

```
package istio.authz
import input.attributes.request.http as http_request

default allow = false

# Get username from input
user_name = parsed {
    [_, encoded] := split(http_request.headers.authorization, " ")
    [parsed, _] := split(base64url.decode(encoded), ":")
}

# RBAC user-role assignments
user_roles = {
    "owner": ["owner"],
    "vfx-1": ["vfx-1"],
    "vfx-2": ["vfx-2"],
    "vfx-3": ["vfx-3"],
    "color": ["color"],
    "sound": ["sound"],
    "hdr": ["hdr"]
}

# RBAC role-permissions assignments
role_permissions = {
    "owner": [{"method": "POST",  "path": "/api/vfx-1"}],
    "vfx-1": [{"method": "POST",  "path": "/api/vfx-2"},
              {"method": "POST",  "path": "/api/vfx-3"}],
    "vfx-2": [{"method": "POST",  "path": "/api/color"}],
    "vfx-3": [{"method": "POST",  "path": "/api/sound"}],
    "color": [{"method": "POST",  "path": "/api/hdr"}],
    "hdr": [{"method": "POST",  "path": "/api/owner"}],
    "sound": [{"method": "POST",  "path": "/api/owner"}]
}

# Logic that implements RBAC
rbac_logic {
    # lookup the list of roles for the user
    roles := user_roles[user_name]
    # for each role in that list
    r := roles[_]
    # lookup the permissions list for role r
    permissions := role_permissions[r]
    # for each permission
    p := permissions[_]
    # check if the permission granted to r matches the user's request
    p == {"method": http_request.method, "path": http_request.path}
}


# ABAC user attributes (tenure)
user_attributes = {
    "owner": {"tenure": 8},
    "vfx-1": {"tenure": 3},
    "vfx-2": {"tenure": 12},
    "vfx-3": {"tenure": 7},
    "color": {"tenure": 3},
    "sound": {"tenure": 4},
    "hdr": {"tenure": 5},
}
```

Listing A.1: Proof of concept access control policy. This policy implements the permissions described in Table 3.1.

```



allow {
  user_name == "owner"

  # Match method and path (RBAC)
  rbac_logic
}

allow {
  user_name == "vfx-1"

  # Match method and path (RBAC)
  rbac_logic
}

allow {
  user_name == "vfx-2"

  # Match method and path (RBAC)
  rbac_logic

  # Match user attributes (ABAC)
  user:=user_attributes[user_name]
  user.tenure > 10
}

allow {
  user_name == "vfx-2"

  # Match method and path (RBAC)
  rbac_logic

  current_time := time.clock([time.now_ns(), "Europe/Paris"])
  to_number(current_time[0]) >= 8
  to_number(current_time[0]) <= 17
}

allow {
  user_name == "vfx-3"

  # Match method and path (RBAC)
  rbac_logic

  # Match user attributes (ABAC)
  user:=user_attributes[user_name]
  user.tenure > 10
}

allow {
  user_name == "vfx-3"

  # Match method and path (RBAC)
  rbac_logic

  current_time := time.clock([time.now_ns(), "Europe/Paris"])
  to_number(current_time[0]) >= 8
  to_number(current_time[0]) <= 17
}

allow {
  user_name == "color"

  # Match method and path (RBAC)
  rbac_logic

  current_time := time.clock([time.now_ns(), "Europe/Paris"])
  to_number(current_time[0]) <= 8
  to_number(current_time[0]) >= 17
}

allow {
  user_name == "sound"

  # Match method and path (RBAC)
  rbac_logic

  current_time := time.clock([time.now_ns(), "Europe/Paris"])
  to_number(current_time[0]) <= 8
  to_number(current_time[0]) >= 17
}

allow {
  user_name == "hdr"

  # Match method and path (RBAC)
  rbac_logic

  current_time := time.clock([time.now_ns(), "Europe/Paris"])
  to_number(current_time[0]) >= 8
  to_number(current_time[0]) <= 17
}
```

A.2 Identity and authentication bootstrap

Since we use a service mesh, services are provided with an identity (certificate) which is associated with a key pair. This identity and this key pair are then used to provide data security at rest, and data security in transport (authentication and encryption) via `mTLS`. Trust in this identity, and the associated authentication mechanism is built on multiple layers, from the service mesh level down to the platform our infrastructure is deployed on. In other words, secure communications at the service mesh level are securely bootstrapped by relying on secure communications at the orchestrator level, which are themselves securely bootstrapped by relying on secure communications at the platform level.

The first step towards trusting the identity and authentication mechanisms is the bootstrap of the orchestrator (Fig. A.1). In the case of Kubernetes [24], processes called kubelets will start running on the orchestrated machines. The kubelets are managed by the node controller. The node controller runs inside the environment of the owner, whereas the kubelets run on all the machines part of the workflow. First, the kubelet will look for its configuration file containing a key pair and a signed certificate. The key pair and signed certificate are subsequently used to enable TLS communications from the kubelet to the node controller. If the kubelet finds it, it can begin normal operation, but if the configuration file does not exist, the kubelet will look for a bootstrap file.

The bootstrap file contains the URL of the node controller, and a limited usage token used to authenticate the kubelet with the node controller (Step 1 of Fig. A.1). After authenticating to the API server with the token, the kubelet obtains limited credentials (Step 2) to create a Certificate Signing Request (CSR). The kubelet sends the CSR to the kube-controller-manager (Step 3). The kube-controller-manager can then automatically approve the CSR, or an outside process, possibly a person, can approve the CSR via the Kubernetes API. Once the CSR is approved, a Certificate is created and issued to the kubelet. The kube-controller-manager can then send the created certificate to the kubelet (Step 4). The kubelet can then create a proper configuration file with the key and signed certificate and can begin normal operation. Optionally, the kubelet can automatically request renewals of the certificate, which goes through the same process as described above.

The orchestrator can now start running the pods on the workers. Kubernetes will use the previously established secure communication channel

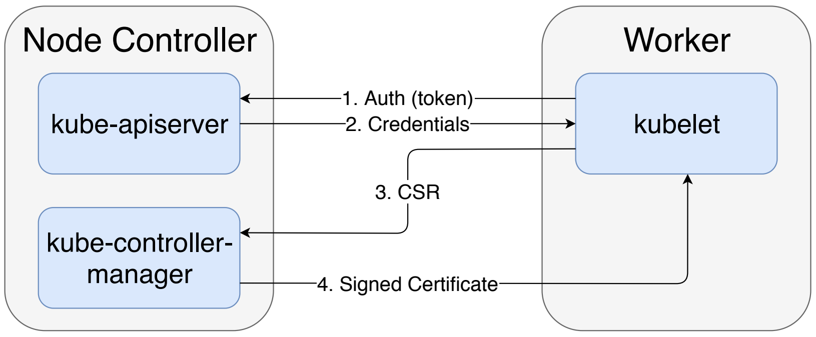


Figure A.1: Kubernetes TLS bootstrap communications. The kubelet uses a limited usage token to authenticate with the API server (Step 1). After authenticating to the API server with the token, the kubelet obtains limited credentials (Step 2) to create a Certificate Signing Request (CSR). The kubelet sends the CSR to the kube-controller-manager (Step 3). The kube-controller-manager can then automatically approve the CSR, or an outside process, possibly a person, can approve the CSR via the Kubernetes API. The kube-controller-manager can then send the created certificate to the kubelet (Step 4), and the kubelet can begin normal operation.

with the kubelet to give the kubelet the YAML configuration file of the pods. The Container Runtime Interface (CRI) on each worker will then make system calls to build the required namespaces until the PID namespace. At this point, the Istio [25] web hook admission controller adds the init containers and the proxy sidecars to the pods, and the Open Policy Agent [27] web hook adds the policy sidecars. The created container images are then started.

Now that kubelets can communicate securely with the node controller and that the pods are running on the orchestrated machines, we need to bootstrap the security of communications between the proxies of our service mesh. The proxies need to setup their key pair and associated certificate to start communicating in the service mesh via `mTLS`. To enable this, a Node Agent running on each worker can receive identity requests from proxies. The proxy sends a JSON Web Token (JWT) along the request to authenticate it (Fig. A.2). When the Node Agent receives the request, it generates a key pair and a CSR, and sends the CSR along with the JWT to the Citadel module of Istio (Fig. A.3). Citadel then authenticates the request and signs the CSR to generate the certificate. Once the Node Agent receives the signed certificate, it sends it along with the key pair to the proxy that requested the identity (Fig. A.4). To achieve certificate and key rotation, this process repeats periodically.

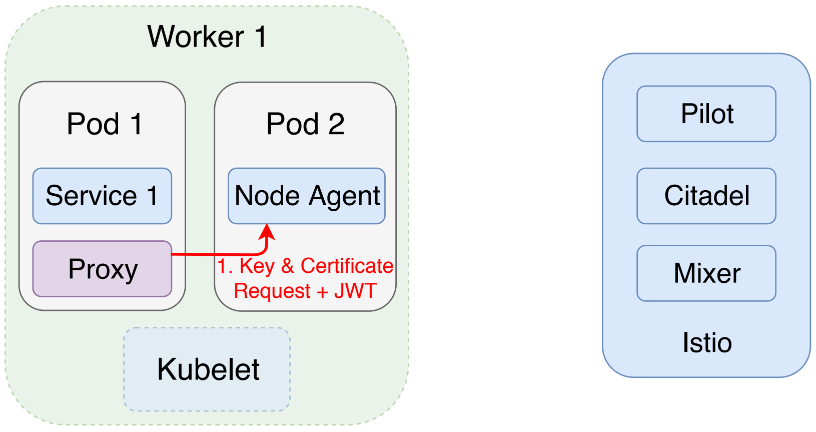


Figure A.2: Istio key distribution via Node Agent: step A. The proxy sends a JWT along an identity request to authenticate it.

Additionally, for the proxies to know which Identity is authorized to run which service, a secure naming information containing this mapping is automatically created from the information retrieved at the Kubernetes node controller. This is sent by the Pilot module of Istio to each proxy. With this bootstrap, each proxy can communicate with the others proxies

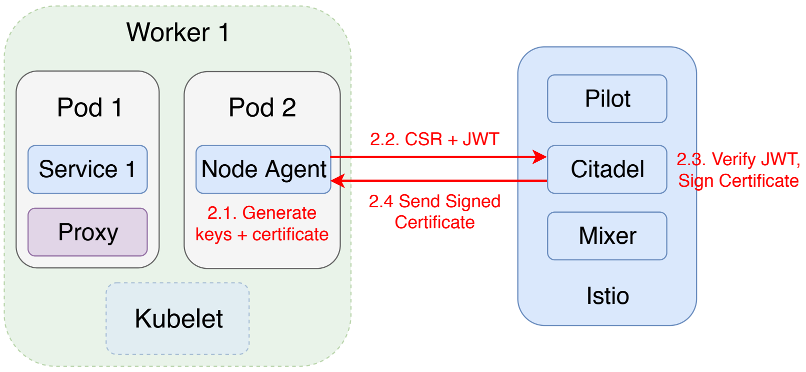

Worker 1
Pod 1
Service 1
Proxy
Pod 2
Node Agent
2.1. Generate keys + certificate
Kubelet
2.2. CSR + JWT
2.4 Send Signed Certificate
Pilot
Citadel
2.3. Verify JWT, Sign Certificate
Mixer
Istio

Figure A.3: Istio key distribution via Node Agent: step B. When the Node Agent receives the proxy's request, it generates a key pair and a CSR, and sends the CSR along with the JWT to the Citadel module of Istio. Citadel then authenticates the request and signs the CSR to generate the certificate.

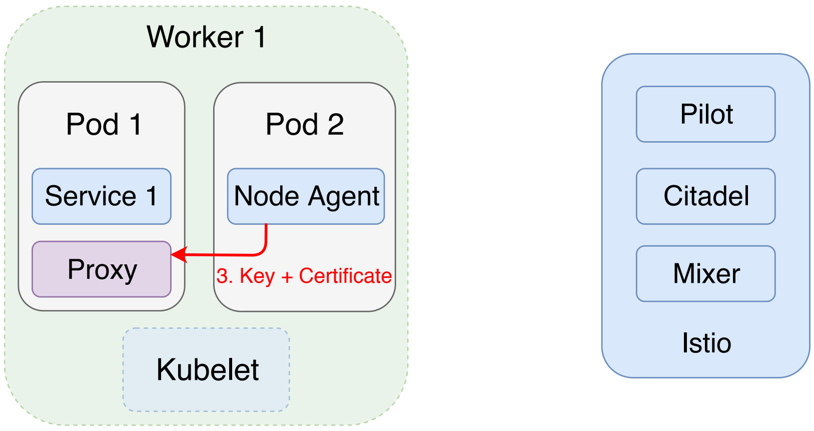


Figure A.4: Istio key distribution via Node Agent: step C. The Node Agent receives the signed certificate and sends it along with the key pair to the proxy that requested the identity.

via `mTLS`, by using their keys to authenticate to each other, and encrypt communications.

Bibliography

Here are the references in citation order.

[1] Igor Polyantchikov et al. 'Virtual enterprise formation in the context of a sustainable partner network'. In: *Industrial management & data systems* (2017) (cited on page 1).

[2] Luis M Camarinha-Matos. *Virtual Enterprises and Collaborative Networks: IFIP 18th World Computer Congress TC5/WG5. 5—5th Working Conference on Virtual Enterprises 22–27 August 2004 Toulouse, France*. Vol. 149. Springer Science & Business Media, 2004 (cited on page 1).

[3] Emilio Esposito and Pietro Evangelista. 'Investigating virtual enterprise models: literature review and empirical findings'. In: *International Journal of Production Economics* 148 (2014), pp. 145–157 (cited on page 1).

[4] Antonio P Volpentesta. 'Hypernetworks in a directed hypergraph'. In: *European Journal of Operational Research* 188.2 (2008), pp. 390–405 (cited on pages 1, 29, 68, 69, 74, 75).

[5] European Commission et al. *Study on data sharing between companies in Europe*. Publications Office, 2018 (cited on page 1).

[6] Alena Lifar et al. *Data Leak Prevention*. 2017. URL: https://datatracker.ietf.org/meeting/100/materials/slides-100-hackathon-sessa-data-leak-prevention-00 (visited on 01/19/2022) (cited on pages 2, 3, 1, 4).

[7] Brian Krebs. *First American Financial Corp. Leaked Hundreds of Millions of Title Insurance Records*. 2019. URL: https://krebsonsecurity.com/2019/05/first-american-financial-corp-leaked-hundreds-of-millions-of-title-insurance-records/ (visited on 02/28/2020) (cited on page 2).

[8] Jonathan Stempel and Jim Finkle. *Yahoo says all three billion accounts hacked in 2013 data theft*. 2017. URL: https://www.reuters.com/article/us-yahoo-cyber/yahoo-says-all-three-billion-accounts-hacked-in-2013-data-theft-idUSKCN1C8201 (visited on 02/28/2020) (cited on page 2).

[9] Tara Seals. *Thousands of MikroTik Routers Hijacked for Eavesdropping*. 2018. URL: https://threatpost.com/thousands-of-mikrotik-routers-hijacked-for-eavesdropping/137165/ (visited on 02/28/2020) (cited on page 2).

[10] Colin Lecher. *Google reportedly fires staffer in media leak crackdown*. 2019. URL: https://www.theverge.com/2019/11/12/20962028/google-staff-firing-media-leak-suspension-employee-termination (visited on 01/15/2020) (cited on pages 2, 3).

[11] OWASP. *OWASP Top Ten*. 2021. URL: https://owasp.org/www-project-top-ten/ (visited on 01/05/2022) (cited on pages 2, 3).

[12] OWASP. *A01:2021 – Broken Access Control*. 2021. URL: https://owasp.org/Top10/A01_2021-Broken_Access_Control/ (visited on 01/05/2022) (cited on pages 2, 3).

[13] Tenable Research. *Tenable Research Reveals Over 40 Billion Records Were Exposed in 2021*. Jan. 2022. URL: https://www.tenable.com/press-releases/tenable-research-reveals-over-40-billion-records-were-exposed-in-2021 (visited on 01/20/2022) (cited on pages 3, 4).

[14] Risk Based Security. *Data Breach Quickview 2020 Year End Report*. 2021. URL: https://pages.riskbasedsecurity.com/en/en/2020-yearend-data-breach-quickview-report (visited on 01/20/2022) (cited on pages 3, 4).

[15] Risk Based Security. *Data Breach Quickview 2019 Year End Report*. 2020. URL: https://pages.riskbasedsecurity.com/2019-year-end-data-breach-quickview-report (visited on 01/20/2022) (cited on pages 3, 4).

[16] Risk Based Security. *Data Breach Quickview Report: An Executive's Guide to Data Breach Trends in 2012*. 2013. URL: https://www.riskbasedsecurity.com/reports/2012-DataBreachQuickView.pdf (visited on 01/20/2022) (cited on pages 3, 4).

[17] Simon Byers et al. 'Analysis of security vulnerabilities in the movie production and distribution process'. In: *Proceedings of the 3rd ACM workshop on Digital rights management*. ACM. 2003, pp. 1–12 (cited on pages 3, 6, 4, 16, 33).

[18] Privacy Rights Clearinghouse. *Data Breaches*. 2020. URL: https://privacyrights.org/data-breaches (visited on 02/28/2020) (cited on pages 3, 4).

[19] Loïc Miller et al. 'Towards Secure and Leak-Free Workflows Using Microservice Isolation'. In: *2021 IEEE 22nd International Conference on High Performance Switching and Routing (HPSR)*. IEEE. 2021, pp. 1–5. DOI: 10.1109/HPSR52026.2021.9481820 (cited on pages 5, 8, 9, 44, 89).

[20] Loïc Miller et al. 'Securing Workflows Using Microservices and Metagraphs'. In: *Electronics* 10.24 (2021), p. 3087 (cited on pages 5, 8, 9, 44, 59, 89, 90).

[21] Loïc Miller et al. 'Verification of Cloud Security Policies'. In: *2021 IEEE 22nd International Conference on High Performance Switching and Routing (HPSR)*. IEEE. 2021, pp. 1–5. DOI: 10.1109/HPSR52026.2021.9481870 (cited on pages 5, 9, 59, 90).

[22] Reynaldo Gil Pons, Max Ward, and Loïc Miller. *Finding (s,d)-Hypernetworks in F-Hypergraphs is NP-Hard*. 2022 (cited on pages 5, 9, 75, 88, 90).

[23] Docker. *Docker*. 2022. URL: https://www.docker.com/ (visited on 02/20/2022) (cited on pages 7, 20, 38).

[24] Kubernetes. *Kubernetes*. 2022. URL: https://kubernetes.io/ (visited on 02/20/2022) (cited on pages 7, 20, 38, 99).

[25] Istio. *Istio*. 2020. URL: https://istio.io/ (visited on 02/28/2020) (cited on pages 7, 21, 38, 100).

[26] Envoy. *Envoy*. 2022. URL: https://www.envoyproxy.io/ (visited on 02/20/2022) (cited on pages 7, 38).

[27] Open Policy Agent. *Open Policy Agent*. 2022. URL: https://www.openpolicyagent.org/ (visited on 02/20/2022) (cited on pages 7, 22, 38, 51, 100).

[28] Dinesha Ranathunga, Matthew Roughan, and Hung Nguyen. 'Verifiable Policy-Defined Networking using Metagraphs'. In: *IEEE Transactions on Dependable and Secure Computing* (2020) (cited on pages 9, 10, 24, 27, 30, 31, 47, 50, 56, 61, 63, 64, 83–85, 87).

[29] OWASP. *A02:2021 – Cryptographic Failures*. 2021. URL: https://owasp.org/Top10/A02_2021-Cryptographic_Failures/ (visited on 01/05/2022) (cited on page 3).

[30] Isaac Manzanera. *Privacy Rights Clearing House Analysis*. Feb. 2016. URL: https://rpubs.com/imanes/168622 (visited on 02/28/2020) (cited on page 4).

[31] hostingtribunal. *Cloud Adoption Statistics for 2021*. Aug. 2021. URL: https://hostingtribunal.com/blog/cloud-adoption-statistics/ (visited on 01/20/2022) (cited on page 4).

[32] Manish Mehta and Torin Sandall. *How Netflix Is Solving Authorization Across Their Cloud*. https://www.youtube.com/watch?v=R6tUNpRpdnY. Dec. 2017 (cited on pages 4, 51).

[33] Vladimir Sumina. *26 Cloud Computing Statistics, Facts & Trends for 2022*. Nov. 2021. URL: https://www.cloudwards.net/cloud-computing-statistics/ (visited on 01/20/2022) (cited on pages 4, 5).

[34] Jacquelyn Bulao. *How Much Data Is Created Every Day in 2021*. Jan. 2022. URL: https://techjury.net/blog/how-much-data-is-created-every-day/ (visited on 01/20/2022) (cited on page 5).

[35] Mike Loukides and Steve Swoyer. *Microservices Adoption in 2020*. 2020. URL: https://www.oreilly.com/radar/microservices-adoption-in-2020/ (visited on 02/14/2022) (cited on page 5).

[36] Cheng Jin, Abhinav Srivastava, and Zhi-Li Zhang. 'Understanding security group usage in a public iaas cloud'. In: *IEEE INFOCOM 2016-The 35th Annual IEEE International Conference on Computer Communications*. IEEE. 2016, pp. 1–9 (cited on page 6).

[37] Ramaswamy Chandramouli and Zack Butcher. *Building Secure Microservices-based Applications Using Service-Mesh Architecture*. Tech. rep. National Institute of Standards and Technology, 2020 (cited on pages 6, 19, 21, 22).

[38] Evan Gilman and Doug Barth. *Zero Trust Networks*. O'Reilly Media, Incorporated, 2017 (cited on page 6).

[39] Mohammadreza Hazhirpasand Barkadehi et al. 'Authentication systems: A literature review and classification'. In: *Telematics and Informatics* (2018) (cited on page 7).

[40] Joseph Bonneau et al. 'The quest to replace passwords: A framework for comparative evaluation of web authentication schemes'. In: *2012 IEEE Symposium on Security and Privacy*. IEEE. 2012, pp. 553–567 (cited on page 7).

[41] Jill Huettich. *What is a workflow? Types, benefits, and examples*. 2020. URL: https://blog.mindmanager.com/blog/2020/06/02/202006202005your-guide-to-the-different-types-of-workflows/ (visited on 02/14/2022) (cited on page 15).

[42] Arthur HM Ter Hofstede et al. *Modern Business Process Automation: YAWL and its support environment*. Springer Science & Business Media, 2009 (cited on pages 16, 17).

[43] Wil MP van der Aalst et al. 'Patterns and XPDL: A critical evaluation of the XML process definition language'. In: *BPM Center report BPM-03-09, BPMcenter. org* (2003), pp. 1–30 (cited on page 16).

[44] OASIS. *Web Services Business Process Execution Language Version 2.0*. 2007. URL: https://docs.oasis-open.org/wsbpel/2.0/OS/wsbpel-v2.0-OS.html (visited on 02/16/2022) (cited on page 16).

[45] Business Process Model. 'Notation (bpmn) version 2.0'. In: *OMG Specification, Object Management Group* (2011), pp. 22–31 (cited on page 16).

[46] Object Management Group. *Unified Modeling Language*. 2017. URL: https://www.omg.org/spec/UML/ (visited on 02/16/2022) (cited on page 16).

[47] Wil MP Van Der Aalst and Arthur HM Ter Hofstede. 'YAWL: yet another workflow language'. In: *Information systems* 30.4 (2005), pp. 245–275 (cited on pages 17, 47).

[48] YAWL Foundation. *How does YAWL compare to BPMN, BPEL, UML and EPCs?* 2022. URL: http://www.yawlfoundation.org/pages/support/faq.html (visited on 02/16/2022) (cited on page 17).

[49] Business Process Management (BPM) Group. *YAWL4Film*. 2010. URL: http://yawlfoundation.org/pages/casestudies/yawl4film.html (visited on 04/14/2021) (cited on pages 18, 47, 86, 87).

[50] Pooyan Jamshidi et al. 'Microservices: The journey so far and challenges ahead'. In: *IEEE Software* 35.3 (2018), pp. 24–35 (cited on page 19).

[51] Canonical. *Linux containers*. 2022. URL: https://linuxcontainers.org/ (visited on 02/20/2022) (cited on page 20).

[52] HashiCorp. *Nomad*. 2022. URL: https://nomadproject.io/ (visited on 02/20/2022) (cited on page 20).

[53] Buoyant. *Linkerd*. 2022. URL: https://linkerd.io/ (visited on 02/20/2022) (cited on page 21).

[54] Ramaswamy Chandramouli. *Security Strategies for Microservices-based Application Systems*. Tech. rep. 2019 (cited on page 22).

[55] Amine El Malki and Uwe Zdun. 'Guiding Architectural Decision Making on Service Mesh Based Microservice Architectures'. In: *European Conference on Software Architecture*. Springer. 2019, pp. 3–19 (cited on page 22).

[56] Murugiah Souppaya, John Morello, and Karen Scarfone. *Application Container Security Guide (2nd Draft)*. Tech. rep. National Institute of Standards and Technology, 2017 (cited on page 22).

[57] Ramaswamy Chandramouli and Ramaswamy Chandramouli. *Security assurance requirements for linux application container deployments*. US Department of Commerce, National Institute of Standards and Technology, 2017 (cited on page 22).

[58] Catherine de Weever and Marios Andreou. 'Zero Trust Network Security Model in containerized environments'. In: (2020) (cited on page 22).

[59] Fatima Hussain et al. 'Intelligent Service Mesh Framework for API Security and Management'. In: *2019 IEEE 10th Annual Information Technology, Electronics and Mobile Communication Conference (IEMCON)*. IEEE. 2019, pp. 0735–0742 (cited on page 23).

[60] Zirak Zaheer et al. 'eZTrust: Network-Independent Zero-Trust Perimeterization for Microservices'. In: *Proceedings of the 2019 ACM Symposium on SDN Research*. 2019, pp. 49–61 (cited on page 23).

[61] Rafael Accorsi and Claus Wonnemann. 'Strong non-leak guarantees for workflow models'. In: *Proceedings of the 2011 ACM Symposium on Applied Computing*. 2011, pp. 308–314 (cited on page 23).

[62] Freeha S Khan et al. 'Data Breach Risks and Resolutions: A Literature Synthesis'. In: (2019) (cited on page 23).

[63] Xiaokui Shu and Danfeng Daphne Yao. 'Data leak detection as a service'. In: *International Conference on Security and Privacy in Communication Systems*. Springer. 2012, pp. 222–240 (cited on page 23).

[64] Mohammad Farhatullah. 'ALP: An authentication and leak prediction model for Cloud Computing privacy'. In: *2013 3rd IEEE International Advance Computing Conference (IACC)*. IEEE. 2013, pp. 48–51 (cited on page 23).

[65] Xiaokui Shu, Danfeng Yao, and Elisa Bertino. 'Privacy-preserving detection of sensitive data exposure'. In: *IEEE transactions on Information Forensics and Security* 10.5 (2015), pp. 1092–1103 (cited on page 23).

[66] Fang Liu et al. 'Privacy-preserving scanning of big content for sensitive data exposure with MapReduce'. In: *Proceedings of the 5th ACM Conference on Data and Application Security and Privacy*. 2015, pp. 195–206 (cited on page 23).

[67] Xiaokui Shu et al. 'Fast detection of transformed data leaks'. In: *IEEE Transactions on Information Forensics and Security* 11.3 (2015), pp. 528–542 (cited on page 23).

[68] Xiaokui Shu et al. 'Rapid screening of transformed data leaks with efficient algorithms and parallel computing'. In: *Proceedings of the 5th ACM Conference on Data and Application Security and Privacy*. 2015, pp. 147–149 (cited on page 23).

[69] Thierry LeVasseur and Philippe Richard. *Data leak protection system and processing methods thereof*. US Patent 9,754,217. Sept. 2017 (cited on page 23).

[70] Carlos Segarra et al. 'Using trusted execution environments for secure stream processing of medical data'. In: *IFIP International Conference on Distributed Applications and Interoperable Systems*. Springer. 2019, pp. 91–107 (cited on page 23).

[71] Chaoshun Zuo, Zhiqiang Lin, and Yinqian Zhang. 'Why does your data leak? Uncovering the data leakage in cloud from mobile apps'. In: *2019 IEEE Symposium on Security and Privacy (SP)*. IEEE. 2019, pp. 1296–1310 (cited on page 23).

[72] Xiangyu Liu et al. 'An empirical study on android for saving non-shared data on public storage'. In: *IFIP International Information Security and Privacy Conference*. Springer. 2015, pp. 542–556 (cited on page 23).

[73] Amiangshu Bosu et al. 'Collusive data leak and more: Large-scale threat analysis of inter-app communications'. In: *Proceedings of the 2017 ACM on Asia Conference on Computer and Communications Security*. 2017, pp. 71–85 (cited on page 23).

[74] Dorothy E Denning. 'A lattice model of secure information flow'. In: *Communications of the ACM* 19.5 (1976), pp. 236–243 (cited on page 23).

[75] Ravi S. Sandhu. 'Lattice-based access control models'. In: *Computer* 26.11 (1993), pp. 9–19 (cited on page 23).

[76] Ravi S Sandhu and Pierangela Samarati. 'Access control: principle and practice'. In: *IEEE communications magazine* 32.9 (1994), pp. 40–48 (cited on page 23).

[77] Ferraiolo David and Kuhn Richard. 'Role-based access controls'. In: *Proceedings of 15th NIST-NCSC National Computer Security Conference*. Vol. 563. Baltimore, Maryland: NIST-NCSC. 1992 (cited on page 24).

[78] ANSI INCITS. 'Incits 359-2012'. In: *Information Technology-Role Based Access Control* (2012) (cited on page 24).

[79] Vincent C Hu et al. 'Guide to attribute based access control (abac) definition and considerations (draft)'. In: *NIST special publication* 800.162 (2013) (cited on page 24).

[80] D. R. Kuhn, E. J. Coyne, and T. R. Weil. 'Adding Attributes to Role-Based Access Control'. In: *Computer* 43.6 (June 2010), pp. 79–81. DOI: 10.1109/MC.2010.155 (cited on page 24).

[81] E. Coyne and T. R. Weil. 'ABAC and RBAC: Scalable, Flexible, and Auditable Access Management'. In: *IT Professional* 15.3 (May 2013), pp. 14–16. DOI: 10.1109/MITP.2013.37 (cited on page 24).

[82] Fulvio Valenza et al. 'Classification and analysis of communication protection policy anomalies'. In: *IEEE/ACM Transactions on Networking* 25.5 (2017), pp. 2601–2614 (cited on page 24).

[83] Karthick Jayaraman et al. 'Automatic error finding in access-control policies'. In: *Proceedings of the 18th ACM conference on Computer and communications security*. 2011, pp. 163–174 (cited on page 24).

[84] Assadarat Khurat, Boontawee Suntisrivaraporn, and Dieter Gollmann. 'Privacy policies verification in composite services using OWL'. In: *Computers & Security* 67 (2017), pp. 122–141 (cited on page 24).

[85] Hongxin Hu, Gail-Joon Ahn, and Ketan Kulkarni. 'Discovery and resolution of anomalies in web access control policies'. In: *IEEE transactions on dependable and secure computing* 10.6 (2013), pp. 341–354 (cited on page 24).

[86] Manuel Koch, Luigi V Mancini, and Francesco Parisi-Presicce. 'Conflict detection and resolution in access control policy specifications'. In: *International Conference on Foundations of Software Science and Computation Structures*. Springer. 2002, pp. 223–238 (cited on page 24).

[87] Fred B Schneider. 'Enforceable security policies'. In: *ACM Transactions on Information and System Security (TISSEC)* 3.1 (2000), pp. 30–50 (cited on page 24).

[88] Manuel Cheminod et al. 'Toward attribute-based access control policy in industrial networked systems'. In: *2018 14th IEEE International Workshop on Factory Communication Systems (WFCS)*. IEEE. 2018, pp. 1–9 (cited on page 24).

[89] Cataldo Basile et al. 'Assessing network authorization policies via reachability analysis'. In: *Computers & Electrical Engineering* 64 (2017), pp. 110–131 (cited on page 24).

[90] Mohsen Rezvani et al. 'Analyzing XACML policies using answer set programming'. In: *International Journal of Information Security* 18.4 (2019), pp. 465–479 (cited on page 24).

[91] Hasiba Ben Attia et al. 'Using Hierarchical Timed Coloured Petri Nets in the formal study of TRBAC security policies'. In: *International Journal of Information Security* 19.2 (2020), pp. 163–187 (cited on page 24).

[92] Alex X Liu et al. 'Xengine: a fast and scalable XACML policy evaluation engine'. In: *ACM SIGMETRICS Performance Evaluation Review* 36.1 (2008), pp. 265–276 (cited on page 25).

[93] Alex X Liu et al. 'Designing fast and scalable XACML policy evaluation engines'. In: *IEEE Transactions on Computers* 60.12 (2010), pp. 1802–1817 (cited on page 25).

[94] Jonathan D Moffett and Morris S Sloman. 'Policy hierarchies for distributed systems management'. In: *IEEE Journal on Selected Areas in Communications* 11.9 (1993), pp. 1404–1414 (cited on page 25).

[95] Arosha K Bandara et al. 'A goal-based approach to policy refinement'. In: *Proceedings. Fifth IEEE International Workshop on Policies for Distributed Systems and Networks, 2004. POLICY 2004.* IEEE. 2004, pp. 229–239 (cited on page 25).

[96] Javier Rubio-Loyola et al. 'A methodological approach toward the refinement problem in policy-based management systems'. In: *IEEE Communications Magazine* 44.10 (2006), pp. 60–68 (cited on page 25).

[97] Robert Craven et al. 'Decomposition techniques for policy refinement'. In: *2010 International Conference on Network and Service Management*. IEEE. 2010, pp. 72–79 (cited on page 25).

[98] Cristian Cleder Machado et al. 'Towards SLA policy refinement for QoS management in software-defined networking'. In: *2014 IEEE 28th International Conference on Advanced Information Networking and Applications*. IEEE. 2014, pp. 397–404 (cited on page 25).

[99] Cataldo Basile et al. ‘A novel approach for integrating security policy enforcement with dynamic network virtualization’. In: *Proceedings of the 2015 1st IEEE Conference on Network Softwarization (NetSoft)*. IEEE. 2015, pp. 1–5 (cited on page 25).

[100] Fulvio Valenza et al. ‘A formal approach for network security policy validation.’ In: *J. Wirel. Mob. Networks Ubiquitous Comput. Dependable Appl.* 8.1 (2017), pp. 79–100 (cited on page 25).

[101] Graham Hughes and Tevfik Bultan. ‘Automated verification of access control policies using a SAT solver’. In: *International journal on software tools for technology transfer* 10.6 (2008), pp. 503–520 (cited on page 25).

[102] Padmalochan Bera, Soumya Kanti Ghosh, and Pallab Dasgupta. ‘Policy based security analysis in enterprise networks: A formal approach’. In: *IEEE Transactions on Network and Service Management* 7.4 (2010), pp. 231–243 (cited on page 25).

[103] Amit Basu and Robert W Blanning. *Metagraphs and their applications*. Vol. 15. Springer Science & Business Media, 2007 (cited on pages 25–27, 30, 31, 47, 48, 50, 63).

[104] Giorgio Gallo et al. ‘Directed hypergraphs and applications’. In: *Discrete applied mathematics* 42.2-3 (1993), pp. 177–201 (cited on pages 28, 29, 31, 66).

[105] Giorgio Ausiello, Paolo Giulio Franciosa, and Daniele Frigioni. ‘Partially dynamic maintenance of minimum weight hyperpaths’. In: *Journal of Discrete Algorithms* 3.1 (2005), pp. 27–46 (cited on page 29).

[106] Mayur Thakur and Rahul Tripathi. ‘Linear connectivity problems in directed hypergraphs’. In: *Theoretical Computer Science* 410.27-29 (2009), pp. 2592–2618 (cited on pages 29, 31).

[107] Giorgio Ausiello, Giuseppe F Italiano, and Umberto Nanni. ‘Hypergraph traversal revisited: Cost measures and dynamic algorithms’. In: *International Symposium on Mathematical Foundations of Computer Science*. Springer. 1998, pp. 1–16 (cited on page 29).

[108] Arindam Mukherjee, Anup Kumar Sen, and Amitava Bagchi. ‘The representation, analysis and verification of business processes: a metagraph-based approach’. In: *Information Technology and Management* 8.1 (2007), pp. 65–81 (cited on page 30).

[109] Ayyoob Hamza et al. ‘Verifying and monitoring iots network behavior using mud profiles’. In: *IEEE Transactions on Dependable and Secure Computing* (2020) (cited on page 31).

[110] Dinesha Ranathunga, Matthew Roughan, and Hung Nguyen. ‘Mathematical reconciliation of medical privacy policies’. In: *ACM Transactions on Management Information Systems (TMIS)* 12.1 (2020), pp. 1–18 (cited on page 31).

[111] D Ranathunga, H Nguyen, and M Roughan. ‘MGtoolkit: A python package for implementing metagraphs’. In: *SoftwareX* 6 (2017), pp. 91–93 (cited on page 31).

[112] Ayyoob Hamza et al. ‘Clear as MUD: generating, validating and applying IoT behavioral profiles’. In: *Proceedings of the 2018 Workshop on IoT Security and Privacy*. 2018, pp. 8–14 (cited on page 31).

[113] Giorgio Ausiello and Luigi Laura. ‘Directed hypergraphs: Introduction and fundamental algorithms—— a survey’. In: *Theoretical Computer Science* 658 (2017), pp. 293–306 (cited on page 31).

[114] Giorgio Ausielloyz et al. ‘Optimal traversal of directed hypergraphs’. In: *ICSI, Berkeley, CA* (1992) (cited on page 31).

[115] Giorgio Ausiello, Paolo G Franciosa, and Daniele Frigioni. ‘Directed hypergraphs: Problems, algorithmic results, and a novel decremental approach’. In: *Italian conference on theoretical computer science*. Springer. 2001, pp. 312–328 (cited on page 31).

[116] Riccardo Cambini, Giorgio Gallo, and Maria Grazia Scutellà. ‘Flows on hypergraphs’. In: *Mathematical Programming* 78.2 (1997), pp. 195–217 (cited on page 31).

[117] James Rumbaugh, Ivar Jacobson, and Grady Booch. ‘The unified modeling language’. In: *Reference manual* (1999) (cited on page 51).

[118] Open Policy Agent. *OPA Adopters*. 2020. URL: https://github.com/open-policy-agent/opa/blob/master/ADOPTERS.md (visited on 11/11/2020) (cited on page 51).

[119] Terence Parr. *The definitive ANTLR 4 reference*. Pragmatic Bookshelf, 2013 (cited on page 54).

[120] Dinesha Ranathunga et al. 'Case studies of scada firewall configurations and the implications for best practices'. In: *IEEE Transactions on Network and Service Management* 13.4 (2016), pp. 871–884 (cited on page 56).

[121] Chris Clifton, Ed Housman, and Arnon Rosenthal. 'Experience with a combined approach to attribute-matching across heterogeneous databases'. In: *Data Mining and Reverse Engineering*. Springer, 1998, pp. 428–451 (cited on page 60).

[122] Seungryong Kim et al. 'Semantic attribute matching networks'. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2019, pp. 12339–12348 (cited on page 60).

[123] Daniele Pretolani. 'Finding hypernetworks in directed hypergraphs'. In: *European Journal of Operational Research* 230.2 (2013), pp. 226–230 (cited on page 68).

[124] Y Shiloach and Y Perl. 'Finding two disjoint paths between two pairs of vertices in a graph'. In: *Journal of the ACM (JACM)* 25.1 (1978), pp. 1–9 (cited on page 69).

[125] Steven Fortune, John Hopcroft, and James Wyllie. 'The directed subgraph homeomorphism problem'. In: *Theoretical Computer Science* 10.2 (1980), pp. 111–121 (cited on page 69).

[126] Stephen A Cook. 'The complexity of theorem-proving procedures'. In: *Proceedings of the third annual ACM symposium on Theory of computing*. 1971, pp. 151–158 (cited on page 71).