# Real-time detection of Advanced Persistent Threats using Information Flow Tracking and Hidden Markov Models

Guillaume Brogi

le cnam

École Doctorale Informatique, Télécommunications et Électronique (Paris)

Centre d'Études et de Recherche en Informatique et Communications

# THÈSE DE DOCTORAT

*présentée par* : **Guillaume Brogi**

*soutenue le* : **4 avril 2018**

*pour obtenir le grade de* : **Docteur du Conservatoire National des Arts et Métiers**

*Spécialité* : **Informatique**

# Real-time detection of Advanced Persistent Threats using Information Flow Tracking and Hidden Markov Models

**Thèse dirigée par**

| | |
|---|---|
| Mme. Di Bernardino Elena | *Maître de Conférences – HDR, Le Cnam* |
| M. Baumard Philippe | *Professeur des Universités – Agrégé des Facultés, Le Cnam* |

**Rapporteurs**

| | |
|---|---|
| Mme. Cénac Peggy | *Maîtresse de Conférences – HDR, Université de Bourgogne* |
| M. Lalande Jean-François | *Maître de Conférences – HDR, CentraleSupélec* |

**Président**

| | |
|---|---|
| M. Mé Ludovic | *Professeur – HDR, CentraleSupélec* |

**Examinateurs**

| | |
|---|---|
| Mme. Viet Triem Tong Valérie | *Professeur-Associé – HDR, CentraleSupélec* |
| M. Zanero Stefano | *Professore Associato, Politecnico di Milano* |
| M. Bar-Hen Avner | *Professeur, Le Cnam* |

# Abstract

In this thesis, we present the risks posed by Advanced Persistent Threats (APTs) and propose a two-step approach for recognising when detected attacks are part of one. This is part of the Akheros solution, a fully autonomous Intrusion Detection System (IDS) being developed in collaboration by three PhD students. The idea is to use machine learning to detect unexpected events and check if they present a security risk. The last part, and the subject of this thesis, is the highlighting of APTs. APT campaigns are particularly dangerous because they are performed by skilled attackers with a precise goal and time and money on their side.

We start with the results from the previous part of the Akheros IDS: a list of events, which can be translated to flows of information, with an indication for events found to be attacks. We find links between attacks using Information Flow Tracking. To do so, we create a new taint for each detected attack and propagate it. Whenever a taint is on the input of an event that is part of another attack, then the two attacks are linked. However, the links are only potential because the events used are not precise enough, which leads to erroneously propagated taints. In the case of an undetected attack, no taint is created for that attack, but the other taints are still propagated as normal so that previous attack is still linked to the next attack, only skipping the undetected one.

The second step of the approach is to filter out the erroneous links. To do so, we use a Hidden Markov Model to represent APTs and remove potential attack campaign that do not fit the model. This is possible because, while each APT is different, they all go through the same phases, which form the hidden states of our model. The visible observations are the kind of attacks performed during these phases. In addition, the results in one phase dictate what the attackers do next, which fits the Markov hypothesis. The score used to rank potential attack campaign from most likely an APT to least likely so is based on a customised Viterbi algorithm in order to take into account potentially undetected attacks.

**Keywords:** Intrusion Detection System, Advanced Persistent Threat, Information Flow Tracking, Hidden Markov Model

# Résumé

Dans cette thèse, nous présentons les risques posés par les Menaces Persistantes Avancées (APTs) et proposons une approche en deux temps pour distinguer les attaques qui en font partie. Ce travail fait partie d'Akheros, un Système de Détection d'Intrusion (IDS) autonome développé par trois doctorants. L'idée est d'utiliser l'apprentissage machine pour détecter des évènements inattendus et vérifier s'ils posent un risque de sécurité. La dernière étape, et le sujet de cette thèse, est de mettre en évidence les APTs. Les campagnes d'APT sont particulièrement dangereuses car les attaquants sont compétents et ont un but précis ainsi que du temps et de l'argent.

Nous partons des résultats des parties précédentes d'Akheros : une liste d'évènements traduisibles en flux d'information et qui indique quand des attaques sont détectées. Nous faisons ressortir les liens entre attaques en utilisant le Suivi de Flux d'Information : nous ajoutons une nouvelle teinte pour chaque attaque. Lors de la propagation, si une teinte se trouve en amont d'un flux qui fait partie d'une attaque, alors les deux attaques sont liées. Certaines attaques se trouvent liées par erreur car les évènements que nous utilisons ne sont pas assez précis, d'où l'approche en deux temps. Dans le cas où certaines attaques ne sont pas détectées, la teinte de cette attaque n'est pas créée, cependant, les autres teintes sont propagées normalement, et l'attaque précédent l'attaque non détectée sera liée à l'attaque lui faisant suite.

Le deuxième temps de l'approche est de retirer les liens erronés. Nous utilisons un Modèle de Markov Caché pour représenter les APTs et retirons les campagnes qui ne suivent pas le modèle. Ceci fonctionne car les APTs, quoique toutes différentes, passent par les mêmes phases. Ces phases sont les états cachés du modèle. Les observations sont les types d'attaques effectuées pendant ces phases. De plus, les actions futures des attaquants dépendent des résultats de l'action en cours, ce qui satisfait l'hypothèse de Markov. Le score utilisé pour classer les campagnes potentielles de la plus proche d'une APT à la plus éloigné est basé sur l'algorithme de Viterbi qui est modifié pour prendre en compte les attaques non détectées potentielles.

**Mots clés :** Système de Détection d'Intrusion, Attaque Persistente Avancée, Suivi de Flux d'Information, Modèle de Markov Caché

# Acknowledgements

First and foremost, I would like to thank Philippe Baumard, who gave me the opportunity to work on this thesis at his company. Without him, this work simply would not exist. At Akheros, I would also like to thank Mark Angoustures and Matthieu Hourbracq, my fellow PhD students. We discussed many ideas over the years, and without them, this work would certainly look very different. Without naming them but they know who they are, the people we met through our industrial partner also deserve thanks. They gave us advice and data making our contributions possible.

Thanks also go to Elena Di Bernardino, who advised me. I am not always easy to manage, but she was always patient and understanding. She guided me in my exploration of Hidden Markov Models, with which I wasn't familiar at all at the beginning of this work, and her insights were crucial in the elaboration of the solution presented here. She was helped in this by Aurélien Latouche, and I would be remiss in not thanking him too. His perpetual joviality, I have taken as encouragement.

I am also very grateful for Valérie Viet Triem Tong and Ludovic Mé. They put me in contact with Philippe in the first place and continued to help and guide me, and they invited me to present my work in front of the CIDRe team multiple time. The rest of the CIDRe team is not forgotten either, always welcoming and encouraging.

Gwladys Médélice and Angélique Grab, the two person in charge of helping students navigate the administration, deserve praise for their dedication to their work. They were always available and prompt to reply, alleviating a large source of stress for us students.

I also want to thank my parents, of course, who started all this and the rest of my family. They have always encouraged me in my studies and have been there for all of my endeavours, including this one. Without them, who knows where I would be today?

And finally, I want to thank my wife, who supported me through this PhD. Her enduring love, as well as her lovely food and cakes, have made this PhD much easier. She is always there when I am tired or when things are difficult. My cats, I am unsure about thanking. They always made sure that I woke early, and that I wouldn't lose sight of the important things in life: their plate`^H^H^H^H^H^H^H` people around us!

# Résumé substantiel

## Détection temps réel de Menaces Persistantes Avancées par Suivi de Flux d'Information et Modèles de Markov Cachés

### Introduction

Les attaques informatiques sont une menace omniprésente qui pèse sur tout système informatique. Les attaques évoluent en même temps que les systèmes ciblés. Elles peuvent avoir pour origine de nombreuses raisons telles que des erreurs de programmation ou de configuration. Il est donc prudent de considérer que tout système informatique est vulnérable, et il faut donc essayer de détecter les attaques qu'il subit. Ceci permet d'arrêter puis d'analyser les attaques afin d'en supprimer la source.

Des Systèmes de Détection d'Intrusion (IDS) sont chargés de détecter ces attaques. Ils peuvent être classés en deux grandes familles : la recherche de signature et la détection d'anomalie. Le principe de la recherche de signature est que l'IDS a une base de données d'attaques, les signatures, et cherche dans le système tout ce qui correspond à une de ces signatures. C'est le fonctionnement des antivirus. L'intérêt de cette méthode est qu'elle est efficace pour détecter des attaques déjà connues. Cependant, elle est incapable de détecter de nouvelles attaques. La méthode peut être adaptée afin de détecter des variantes d'attaques connues, mais ceci augmente la quantité de faux positifs.

À l'inverse, les IDS basés sur la détection d'anomalie ne connaissent rien des attaques. Ils ont au contraire un modèle du système à protéger, et recherchent toute déviation par rapport à ce modèle. L'intérêt de cette approche est qu'elle est capable de détecter des attaques jusqu'alors inconnues puisque l'IDS ne connait aucune attaque. Cependant, cette approche a tendance à présenter de nombreux faux positifs car il est très difficile d'avoir un modèle complet et à jour du système. Du coup, dès que le système exécute une action légitime mais absente du modèle, cette action est vue comme une attaque. Bien sûr, il est possible de relâcher le modèle, mais faire ainsi augmente la possibilité

de faux négatifs puisqu'une attaque qui n'induit qu'une faible déviation par rapport au modèle ne sera pas détectée.

Ainsi, les deux familles d'IDS doivent faire des compromis entre la détection d'autant d'attaques possibles tout en limitant le nombre de faux positifs et de faux négatifs. Ces compromis sont d'autant plus importants qu'un nouveau type d'attaquants est actif. Ces attaquants travaillent souvent pour des entités riches et puissantes telles que des gouvernements ou des multinationales. Leurs campagnes d'attaques sont appelées les Menaces Persistantes Avancées (APTs) et sont en général motivées par l'espionnage et le sabotage. Ces attaquants sont sont capables de créer de nouveaux malwares basés sur de nouvelles vulnérabilités (0-days) ou modifier des malwares existants pour éviter qu'ils soient détectés, et ils peuvent prendre le temps nécessaire pour s'assurer de pénétrer leur cible et atteindre leurs objectifs.

Les caractéristiques principales des APTs sont le fait que les attaquants sont sont capables de développer de nouveaux malwares et que l'attaque est motivée par un but précis et donc ciblée. Contrairement à une attaque opportuniste où l'attaquant cherche un nombre limité de vulnérabilités dans un grand nombre de cibles, les APTs des cibles choisies à l'avance et vont donc prendre le temps d'analyser chaque cible afin d'y trouver des vulnérabilités pour pouvoir pénétrer le système. Cette approche est rendue possible par la capacité des attaquants à modifier des attaques existantes pour éviter leur détection et à développer de nouvelles attaques utilisant des 0-days.

Un exemple récent est la campagne Monsoon, durant laquelle les attaquants prenaient contrôle des machines de leurs victimes avec un mail de phishing. Forcepoint a analysé la campagne dans [Forcepoint 2016] et a trouvé deux nouvelles familles de malware, BADNEWS et TINYTYPHON, qui étaient donc indétectable par des IDS à signature. De plus, ces malwares communiquaient avec leur centre de commande en utilisant des canaux originaux mais difficile à détecter : les champs de commentaires sur Github, des sites d'actualités, etc…

Ces campagnes d'attaques sont donc de réelles menaces et leur détection pose de nouveaux défis. Des idées sont déjà en train d'être développées pour répondre à ces défis, telles que [Vance 2014 ; Chandran, Hrudya et Poornachandran 2015] qui se concentrent sur la détection d'attaques inconnues ou [Giura et W. Wang 2012 ; Sexton, Storlie et Neil 2015] qui tentent détecter des campagnes complètes.

Akheros est une startup créée avec pour but explicite de développer un IDS autonome capable de détecter des APTs. Afin de pouvoir détecter des attaques inconnues, l'IDS s'inspire de ceux basés sur la détection d'anomalie et ne connait donc rien des attaques elles-mêmes. Cependant, pour pouvoir être vraiment autonome, le modèle du système

à protéger n'est pas non plus fourni à l'IDS. Ce développement a été partagé en trois modules qui s'appuient sur les données fournies par de nombreuses sondes qui observent le systèmes à différent niveaux, des appels systèmes aux accès utilisateurs.

Le premier module, chargé de l'apprentissage du système à protéger, est le sujet de la thèse de Matthieu Hourbracq. Il aborde le problème en utilisant un réseau bayésien pour modéliser le comportement du système à un moment donné. Cependant, lorsque le comportement dévie suffisamment du modèle, l'IDS enregistre cette déviation dans un nouveau modèle. Il crée ainsi autant de modèles que nécessaires, et obtient, par exemple, que le système suit un premier modèle puis passe à un deuxième modèle, revient au premier puis par sur un troisième, et ainsi de suite. L'IDS apprend aussi un méta-modèle qui représente les transitions d'un modèle à l'autre. Armés de tous ces modèles, qui évoluent avec le temps et les changements dans le système, l'IDS est capable d'identifier les évènements incongrus, tels que la création d'un nouveau modèle très différent des autres modèles, ou la transition d'un modèle à un autre à un moment inattendu.

Le deuxième module analyse le danger posé par les évènements incongrus mis en avant par le premier module. C'est le sujet de la thèse de Mark Angoustures. Ce module commence par représenter les processus et leurs actions sous forme de graphe afin de localiser précisément la source de l'incongruité. En effet, même si l'incongruité est due à une action effectuée par un premier processus, ce processus peut-être le résultat d'un deuxième processus. Le module va ainsi pouvoir déterminer le processus responsable de l'incongruité. Il va ensuite analyser l'ensemble des actions de ce processus afin de déterminer si ce dernier représente un risque pour la sécurité du système et, le cas échéant, caractériser la nature de ce risque.

Le troisième et dernier module est le sujet de cette thèse. Il est chargé de trouver les liens, s'ils existent, entre les attaques détectées par le deuxième module. Ceci permet de retrouver les campagnes d'attaques complètes afin de donner aux défenseurs un maximum d'information lorsqu'ils doivent répondre à ces attaques. En effet, une attaque peut sembler bénigne, mais si on s'aperçoit qu'elle fait partie d'une campagne d'attaque plus étendue, il faudra alors l'analyser avec plus de soin.

Ce rapport de thèse présente donc mes contributions à cet effort. Elles sont au nombres de trois. La première est l'utilisation du Suivi de Flux d'Information (IFT) afin d'établir des liens entres attaques potentiellement liées. La deuxième est l'établissement d'un Modèle de Markov Caché (HMM) représentant les APTs afin de vérifier que les liens établis par IFT sont raisonnables. Cette partie comprend aussi le développement d'un score permettant de comparer des chaines de différentes longueur et prenant en compte les attaques potentiellement non détectées. La troisième contribution est connexe aux deux premières puisqu'elle consiste à la mise en place d'un outil, Moirai, afin de jouer

des scénarios d'attaque facilement. Ceci permet de tester facilement les deux premières contribution et permet aussi de comparer différents IDS en rendant le partage de scénario aussi simple que possible.

## Suivi de Flux d'Information – Trouver les liens entre attaques

Le concept du Suivi de Flux d'Information (IFT) est simple : l'idée est de suivre le parcours des données au sein d'un système. Les applications sont variées. Par exemple, on peut imaginer que certaines données ne doivent pas apparaitre sur le réseau. Grâce à l'IFT, on peut suivre ces données et bloquer toute tentative de les envoyer sur le réseau sans savoir comment elles pourraient arriver là.

L'IFT peut être implémenté à différents niveaux, chaque niveau ayant des avantages et des inconvénients. [Suh et al. 2004] propose une implémentation hardware, utilisant un CPU modifié. Ce CPU est capable de suivre les flux d'information à travers tout le système et avec une bonne précision. Cependant, les politiques qu'il peut implémenter sont rigides et le déploiement d'un CPU modifié est compliqué. Au contraire, [Qin et al. 2006] propose une implémentation purement software qui instrumente les binaires. Ce genre d'approche est plus facile à déployer, mais ne peut pas suivre les flux d'information entre processus. Le but des deux approches est pour autant similaire puisqu'il s'agit de bloquer l'utilisation de données entrées par l'utilisateur dans le flux d'exécution des processus, c'est à dire les attaques du genre dépassement de pile et de tampon.

Nous utilisons une implémentation entre ces deux niveaux : l'IFT est effectué au niveau de l'OS. Ceci rend possible le suivi des flux entre processus sans pour autant demander de déployer du matériel spécifique. C'est l'approche proposée par [Enck et al. 2014] qui l'utilise sous Android pour empêcher l'utilisation de données privée par des applications non autorisées. [Hauser, Tronel, Reid et al. 2012] étend le suivi des flux au réseau en modifiant Linux. Ceci permet à un parc de machine de collaborer pour suivre les flux et ainsi de proposer une politique de sécurité à l'échelle du réseau complet. Cependant, cette approche n'est pas capable de suivre les flux à l'intérieur d'un processus. Si un processus a deux entrées de flux, toutes les sorties de ce processus devrons considérer qu'elles sont un mélange de ces deux flux, même si ce n'est pas effectivement le cas. Ceci induit des faux positifs. Cependant, nous choisissons quand même de nous positionner à ce niveau car nous avons besoin de suivre les flux entre processus et les sondes dont nous disposons déjà nous apportent toutes les informations nécessaires.

Pour nous, le but de l'IFT est d'établir des liens entre attaques. Pour ce faire, nous suivons donc les flux au niveau des processus, fichiers et sockets. L'idée est de suivre les

données provenant des attaques et de déterminer si les flux propageant ces données font parties d'autres attaques. Pour ce faire, nous commençons par associer un tag à chaque attaque. Ensuite, tous les évènements qui ont été identifiés comme faisant partie d'une attaque sont traduits en flux d'information et propagent le tag de cette attaque. Les flux qui ne font pas parties d'attaques propagent les tags qu'ils ont en entrées vers leurs sorties. Ces tags nous indiquent donc où les données provenant d'attaques sont dispersées. Finalement, nous établissons des liens entre attaques à chaque fois qu'un flux d'information faisant partie d'une attaque a, en entrée, des données associées à un tag d'une autre attaque.

Prenons l'exemple présenté Figure 5.3 page 87. Nous avons une campagne d'APT comportant quatre attaques sur un serveur web et sa base de donnée. Le but est de lier ces quatre attaques avec l'IFT. En même temps que cette campagne, le site web est aussi visité par des utilisateurs normaux et est la cible de deux autres attaques opportunistes. En effet, les campagnes d'APT pouvant durer des mois, il est probable que le même système subira d'autres attaques en simultané.

En utilisant l'IFT, nous lions les deux premières attaques de la campagne d'APT comme présenté Figure 3.5 page 52. Un tag, représenté en bleu, est propagé par le flux faisant partie de la première attaque. Ce tag est propagé par d'autres flux qui nous permettent de suivre l'exécution d'un premier script qui va télécharger un deuxième script caché sous le nom de `sshd` et qui se connecte à IRC pour attendre des instructions. L'attaquant explore le système et initie une deuxième attaque, représentée par le tag rouge. Puisque le flux faisant partie de la deuxième attaque provenait d'une source bleue, les deux attaques sont considérée comme liées. On continue à propager les différents tags et on arrive effectivement à lier les quatre attaques qui font partie de cet APT, comme indiqué Figure 3.6f page 53.

Cependant, on observe aussi que nous avons lié la première attaque de l'APT avec les deux attaques opportunistes. Ceci est dû aux limitations présentée précédemment. Dans ce cas précis, les trois attaques ciblent le même processus, le serveur web. Après la première attaque, ce processus reçoit le tag de cette attaque, et lorsque les autres attaques sont détectées, elles se retrouvent liées à la première. C'est pour cette raison que nous filtrons ensuite les liens avec un Modèle de Markov Caché (HMM).

En conclusion, grâce aux sorties des modules précédents d'Akheros, l'IFT est capable de suivre les flux d'information et de lier les attaques qui font parties d'une même campagne. Il est intéressant de noter que cette méthode fonctionne quel que soit le temps qui s'écoule entre les attaques, ce qui est un avantage pour détecter les APTs, qui sont des campagnes pouvant durer plusieurs mois. Cependant, cette méthode crée aussi des liens qui n'ont pas lieu d'être. Ces liens sont dûs au manque de précision inhérent au suivi

de flux d'information au niveau de l'OS. Le niveau OS est le bon niveau pour pouvoir suivre les flux qui nous intéressent, mais il faut donc, après l'IFT, filtrer les liens établis afin d'éliminer ceux qui sont erronés.

## Modèle de Markov Caché – Évaluation des liens

Les Modèles de Markov Cachés sont des modèles stochastiques à deux niveaux. Le premier niveau est une chaine de Markov, c'est à dire, processus stochastique dont l'état au moment $t$ ne dépend que de l'état au moment $t-1$. Ce processus ne peut pas être mesuré directement. Le deuxième niveau du modèle de Markov est la partie observable. Ce niveau comporte une observation pour chaque moment de la chaine, et l'observation au moment $t$ ne dépend que de l'état au même moment $t$. Un exemple classique d'application des HMMs est la détermination des températures d'années non-mesurées en observant la tailles des cercles de troncs d'arbres pour ces années. Les cercles n'indiquent pas précisémment si une année était chaude ou froide, mais les années froides ont tendance à produires des cercles plus petits, et en observant ces cercles, on peut essayer de retrouver la séquence d'années chaudes et froides.

Formellement, un HMM $\lambda$ est défini par sa liste de $N$ états $S$, sa liste de $M$ états $O$, la matrice de transition d'états $A$, la matrice d'observations $B$ et le vecteur initial $\pi$. Un HMM peut être utilisé pour répondre à trois classes de questions. La première est le calcul du score d'une séquence d'observation pour un modèle donné, c'est à dire savoir si les observations suivent le modèle. La deuxième est de retrouver la séquence d'états cachés optimale pour la séquence d'observations mesurées et pour un modèle donné. La troisième est la détermination du modèle qui correspond à une séquence d'observation. Un certain nombre d'algorithmes sont disponibles pour répondre à chacune de ces questions.

L'utilisation de HMMs n'est pas une chose nouvelle dans le domaine de la sécurité. Ils ont déjà servi à obtenir un modèle du système à protéger, tel que dans [Arnes et al. 2006] où les états du HMM représentent l'état du système et les observations sont les sorties des sondes des IDS. Et ils peuvent aussi servir à modéliser l'attaque, comme dans [Ourston et al. 2003] où les états représentent les étapes d'une attaque. Dans notre cas, le HMM est utilisé pour modéliser le scénario d'attaque afin de classer les différentes chaines d'attaques obtenues par IFT en fonction de leur probabilité d'être effectivement une APT. C'est à dire que nous voulons répondre à la première question proposée plus haut : est-ce que la chaine d'observations correspond au modèle ? À la différence près que nous voulons savoir si la chaine correspond au modèle plus que les autres chaines. Une fois que nous avons répondu à la première question, la réponse à la deuxième question permet aux défenseurs humains de réagir de manière appropriée à l'attaque.

Les scores habituels sont construits pour déterminer si une chaine donnée correspond au modèle mais ne peuvent pas être utilisés pour comparer des chaines de différentes tailles. Par exemple, le Critère d'Information Bayésien (BIC) est défini par $BIC = \ln(L) \cdot k - 2 \cdot \ln(\mathcal{L})$, où $L$ est la longueur de la chaine, $k$ est le nombre de paramètres du modèle et $\mathcal{L}$ est la log-vraisemblance de la chaine : $\mathcal{L} = \ln(\mathbb{P}(o, s|\lambda)) = \ln(\mathbb{P}(o_L|s_L) \cdot \mathbb{P}(s_L|s_{L-1}) \cdots \mathbb{P}(o_1|s_1) \cdot \mathbb{P}(s_1))$. Si on applique le BIC sur les chaines trouvées par IFT dans la Figure 3.6f page 53, on se rend compte que le score de la deuxième chaine est meilleur que celui de la première chaine malgré le fait que cette chaine n'est pas une APT. Ceci est dû au fait que la chaine est plus courte, et donc la log-vraisemblance est meilleure.

Nous devons donc trouver un moyen de rendre le score indépendant de la longueur de la chaine. Pour ce faire, nous commençons par la définition de la log-vraisemblance : $\mathcal{L} = \ln(\mathbb{P}(o_L|s_L) \cdot \mathbb{P}(s_L|s_{L-1}) \cdots \mathbb{P}(o_1|s_1) \cdot \mathbb{P}(s_1))$. On se rend compte qu'on a $2 \cdot L$ multiplications, et chacun des termes est une probabilité dans les matrices $A$, $B$ et $\pi$. On peut donc trouver un encadrement pour la log-vraisemblance, et modifier le terme central pour que les bornes ne dépendent pas de la longueur de la chaine :

$$
\begin{aligned}
2 \cdot L \cdot \ln\left(\min_{ij}(b_{ij}, a_{ij}, \pi_i)\right) \leq & \mathcal{L} \leq 2 \cdot L \cdot \ln\left(\max_{ij}(b_{ij}, a_{ij}, \pi_i)\right) \\
2 \cdot \ln\left(\min_{ij}(b_{ij}, a_{ij}, \pi_i)\right) \leq & \frac{\mathcal{L}}{L} \leq 2 \cdot \ln\left(\max_{ij}(b_{ij}, a_{ij}, \pi_i)\right)
\end{aligned}
$$

On utilise donc $\frac{\mathcal{L}}{L}$ comme base pour notre score et gardons le fait que le score est positif et qu'un plus petit score est meilleur. Nous définissons donc notre score comme : $\mathcal{S} = -\frac{\mathcal{L}}{L}$. Comme on peut le voir Figure 4.7 page 71, le score est indépendant de la longueur de la chaine et est capable de séparer les chaines d'APTs des chaines qui n'en sont pas. Si on revient aux deux chaines du paragraphe précédent, on obtiens bien un meilleur score pour la chaine qui représente l'APT et un score plus mauvais pour celle qui n'en est pas une.

Maintenant que nous avons un score qui permet de comparer des chaines de différentes tailles, nous voulons prendre en compte le fait que certaines attaques ne sont pas forcément détectée. En effet, les attaques effectuées pendant des APTs sont difficile à détecter. Il est donc raisonnable de penser que certaines passeront inaperçues. En terme de HMM, cela veut dire que certaines observations seront manquantes, sans que nous sachions à quel moment elles manquent. Nous voulons donc modifier le score pour prendre

en compte ces observations potentiellement manquantes.

Pour cela, nous commençons par observer que la probabilité de passer de l'état $i$ à l'état $j$ en sautant un état est $\mathbb{P}(s_{t+2} = S_j | s_t = S_i) = \sum_{k=1}^{N} a_{i,k} \cdot a_{k,j}$. Ceci correspond à l'élément $A^2[i,j]$. Nous observons que la matrice $A^2$ est une matrice de transition, c'est à dire que chaque élément de la matrice est dans $[0,1]$ et que chaque ligne somme à 1. Ceci est vrai pour $A^k, \forall k \in \mathbb{N}$. Nous décidons donc d'intégrer ces matrices dans le calcul de la log-vraisemblance avec un poids décidé par la probabilité $p$ de manquer une observation. Nous ne pouvons pas intégrer toutes les matrices pour $k \in \mathbb{N}$ et choisissons donc de restreindre $k$ à $[\![1, K]\!]$, où $K$ est choisi de manière arbitraire afin de limiter le temps de calcul. Le poids associé à la matrice $A^k$ est donc $\frac{p^{k-1}}{\sum_{n=1}^{K} p^{n-1}}$. Il est important de noter que plus $k$ est grand, plus ce poids est petit, et donc le choix de $K$ revient à choisir la précision voulue pour l'approximation que nous faisons. Si nous notons $S_i^{(k)}$ l'état $S_i$ atteint en $k$ étapes, c'est à dire en manquant $k-1$ observation, nous avons donc :

$$\text{pour } i, j \in [\![1, N]\!], \quad k \in [\![1, K]\!], \quad \forall t,$$
$$\mathbb{P}(s_{t+1} = S_j^{(k)} | s_t = S_i) = A^k[i,j] \cdot \frac{p^{k-1}}{\sum_{n=1}^{K} p^{n-1}}$$

Nous pouvons utiliser ces probabilités pour calculer une log-vraisemblance qui prend en compte jusqu'à $K-1$ observations manquantes à la suite, notée $\mathcal{L}^{(K)}$ et l'intégrer à notre score, que nous notons donc $\mathcal{S}^{(K)}$. Figure 4.8 à Figure 4.16 pages 72–76 montrent l'évolution de la classification de chaines quand la probabilité $p$ de ne pas détecter une observation augmente. Nous remarquons que plus $p$ augmente, plus la distinction entre APT et non-APT diminue. Ceci est dû au fait que plus $p$ augmente, moins le score est pénalisé lorsqu'il considère des attaques non-détectées. Une chaine non-APT peut donc être considérée plus facilement comme une APT en rajoutant des étapes manquantes. Nous avons donc bien un score qui est indépendant de la longueur de la chaine et qui prend en compte la probabilité de manquer des observations.

## Moirai – Un outil pour rejouer des scénarios

Cette troisième contribution ne fait pas directement partie de l'IDS. Cependant, son élaboration était nécessaire pour pouvoir évaluer les autres contributions. En effet, l'évaluation d'IDS passe en général par l'utilisation de jeux de données pré-existants. Cependant, ces jeux de données deviennent rapidement obsolètes, comme le montre [Małowidzki, Bereziński et Mazur 2015]. De plus, les jeux de données étant construit indépendam-

ment des IDS, les données qu'ils contiennent ne peuvent pas forcément être utilisées par tous les IDS. Par exemple, la plupart des jeux de données sont soit pour IDS qui observent le réseau ou pour IDS qui observent les hôtes, mais pas pour les deux. Il est donc compliqué de comparer des IDS différents en utilisant ces jeux de données.

De plus, dans notre cas, nous n'avons pas trouvé de jeux de données permettant de faire du suivi de flux d'information et comportant des APTs ainsi que d'autres attaques indépendantes. Nous avons donc créé Moirai. Moirai est un outil qui répond aux problématiques de comparaison d'IDS hétérogènes et qui permet de tenir les jeux de données à jours. Pour ce faire, Moirai ne crée pas de jeu de données. Moirai se situe un niveau plus haut et s'occupe de rejouer les scénarios utilisés pour générer les jeux de données. De plus, une des intentions de Moirai est de faciliter le partage de ces scénarios.

Technologiquement, Moirai est en fait une fine couche au dessus de technologies déjà existantes. Moirai utilise Vagrant [HashiCorp 2010] pour partager et gérer les machines virtuelles (VMs) utilisées par les scénarios. Vagrant permet de démarrer chaque scénario avec des VMs propres et donc de les rejouer à l'identique à chaque fois. Pour se connecter au VMs et effectuer les actions des scénarios, Moirai utilise soit `ssh` pour les VMs UNIX soit `Windows Remote Management` pour les machines Windows.

Les scénarios eux-mêmes sont définis dans de simples fichiers textes au format INI. Ces fichiers définissent chaque VM Vagrant ainsi que le timing de chaque action du scénario. Pour les partager, il suffit donc de partager ce fichier texte ainsi que les logiciels particuliers installés sur les VMs. Pour ce dernier point, nous recommandons de créer des VMs template contenant tous les logiciels nécessaires et de les partager via le site de Vagrant. Pour la personne qui souhaite tester un IDS, il suffit alors de récupérer les bonnes templates de VMs et de les surcharger en installant l'IDS à tester. Ensuite, il suffit d'une commande Moirai pour que le scénario se joue.

Cette approche présente plusieurs intérêts. Tout d'abord, l'IDS à tester tourne directement dans le scénario. Il est donc certain d'avoir toutes les informations dont il a besoin puisqu'il les collecte lui-même. Cela permet donc de comparer des IDS hétérogènes. L'autre avantage est que les scénarios sont facile à comprendre et à modifier. Si un scénario utilise des attaques obsolètes, il suffit de modifier le scénario en changeant ces attaques pour obtenir un scénario à jour. De plus, il est possible d'étendre et de combiner des scénarios. Par exemple, si on a deux scénarios avec une campagne d'attaque chacun, on peut combiner ces scénarios et rajouter des attaques indépendantes. On obtient alors un scénarios plus complexe qui permet de tester les IDS dans des situations plus difficile à analyser.

L'outil est open source [Akheros 2016a] et deux scénarios sont déjà disponibles

[Akheros 2016b]. Ce sont les scénarios que nous avons utilisés pour tester nos autres contributions.

## Conclusion

En conclusion, nous avons montré qu'il est possible de trouver les liens entre attaques afin de reconstruire les campagnes, et en particulier les APTs. Le but indiqué est donc atteint, mais certaines contraintes ont été relâchées. En particulier, le système n'est pas complètement autonome puisqu'il nécessite un modèle de campagne d'attaque.

Nous avons présenté trois contributions majeures. Les deux premières contribuent directement à la détection des campagnes d'attaques tandis que la troisième est nécessaire pour évaluer les deux autres.

La première contribution est l'application du Suivi de Flux d'Information (IFT) pour la mise en évidence des liens entre attaques. Nous n'avons pas apporté de modifications au concept de l'IFT, mais nous démontrons une nouvelle application. Dans nos tests, l'IFT a toujours été capable de lier les attaques faisant partie de la même campagne. Malheureusement, l'IFT lie aussi des attaques qui ne font pas partie de la même campagne, ce qui motive la deuxième contribution. Cette première contribution a été publiée à NTMS en 2016 : [Brogi et Viet Triem Tong 2016].

La deuxième contribution est l'utilisation d'une Chaine de Markov Cachée (HMM) pour modéliser les campagnes d'attaques. Cette contribution est utilisée pour classer les campagnes d'attaques de la plus probablement une APT à la moins probablement une APT. Ceci permet de filtrer les vrais des faux positifs trouvés par l'IFT. Cette contribution montre donc que les HMMs peuvent être utilisées pour modéliser les campagnes d'attaques et introduit un score avec deux particularités. La première particularité est de permettre de comparer des chaines de tailles différentes. Ce besoin est dû au fait que nous comparons des chaines dont nous ne contrôlons pas la taille afin de les classer, alors que l'utilisation classique d'un HMM est de comparer une chaine à un ou plusieurs modèles. Ce qui explique qu'un tel score n'existait pas. La deuxième particularité de notre score est qu'il prend en compte la possibilité que certaines observations manquent. Nous avions besoin d'un tel score car la détection des attaques composant les APTs est difficile et rien ne garantit que toutes les attaques seront détectées. Cette contribution est en cours de publication au IJSN : [Brogi et Di Bernardino 2018].

La troisième contribution est l'élaboration de Moirai, un outil pour rejouer et partager des scénarios d'évaluations d'IDS. La création de cet outil s'est révélée nécessaire devant le manque de jeu de donnée permettant d'évaluer les IDS détectant des APTs. Moirai

permet de définir des scénarios dans un simple ficher texte ce qui les rend facile à partager. L'IDS a tester tourne directement dans l'environnement qu'il est sensé protéger et récupère donc toutes les données dont il a besoin. Cette contribution a été publiée à RESSI en 2017 : [Brogi et Viet Triem Tong 2017]. L'outil lui-même ainsi que les scénarios que nous avons développés sont disponibles sur Github : [Akheros 2016a ; Akheros 2016b].

Bien sûr, ces contributions ne sont pas parfaites. Le plus gros défaut concerne l'évaluation. Malgré la création de Moirai qui rend le rejeu de scénario trivial, la création d'un nouveau scénario prend du temps. Nous n'avons qu'un nombre limité de scénario qui limite donc la qualité de l'évaluation. Nous avons aussi identifié d'autre défauts dans notre approche. Le premier est que nous n'avons pas traité les faux positifs de l'IFT dans l'IFT lui-même. En effet, le HMM est là pour classer les chaines trouvées par l'IFT, et les faux positifs se retrouvent en bas du classement. Cependant, si nous pouvions réduire les faux-positifs au niveau de l'IFT, cela rendrait certainement le classement produit par le HMM plus précis. Le deuxième défaut est dans le HMM. Ce dernier est créé au préalable et n'utilise pas les données traitées au fur et à mesure pour s'améliorer. La mise à jour du modèle doit donc se faire hors-ligne, et demande donc autant de travail que l'élaboration initiale, ce qui est contraire à notre but d'autonomie de l'IDS. Le troisième défaut réside dans le score. En effet, celui-ci permet de prendre en compte les données potentiellement manquante. Les résultats peuvent donc être difficile à interpréter. Dans le cas où nous supprimons nous même des données, ceci est possible. Mais dans le cas où nous ne savons pas si des données sont manquantes, l'évaluation requiert, pour chaque chaine, de regarder à quel endroit le score trouve des données manquantes et d'imaginer quelles données seraient les plus avantageuses pour interpréter cette chaine comme une APT.

Face à ces défauts, nous avons identifié des travaux futurs. Le plus gros et le plus pressant est l'élaboration de nouveaux scénarios, combinant plusieurs APTs. Pour l'instant, les scénarios existants combinent une APT avec des attaques indépendantes et des actions légitimes. Nous souhaitons étudier le comportement de notre approche quand plusieurs APTs sont mêlées. Pour aider dans l'élaboration de ces nouveaux scénarios, il pourrait être utile, mais pas nécessaire, de rajouter de nouvelles capacités à Moirai. En particulier, l'outil devrait être capable de mettre en place un spoof d'IP/DNS configurable. Ceci permettra d'intégrer plus facilement de véritable malware qui ne serait pas forcément configurable.

Un autre point sur lequel nous souhaitons travailler est le traitement des faux positifs au niveau de l'IFT. L'idée principale est d'utiliser la sortie du HMM afin de supprimer certaines teintes. Par exemple, si le HMM indique qu'un attaque fait partie d'une APT et

correspond à la phase d'installation de l'outil d'administration à distance de la campagne d'attaque, les teintes des phases précédentes de la campagne devraient pouvoir être supprimées car les nouvelles attaques devrait logiquement passer par cet outil là. Une autre possibilité est d'utiliser les chaines qui sont vues comme n'étant pas des APTs, mais il faut encore trouver une bonne manière d'utiliser cette information.

Dernièrement, nous souhaitons aussi améliorer le HMM en ligne. Une idée serait d'utiliser une boucle de retour comprenant un humain : après analyse par un humain d'une chaine classée par le HMM, celui-ci peut indiquer si le classement est approprié. Le HMM peut utiliser cette information pour améliorer son modèle. Une autre possibilité à explorer, lorsque le modèle sera capable de s'améliorer en ligne, est l'élaboration d'un modèle d'HMM par groupe effectuant des APTs. En effet, les rapports sur les APTs identifient les groupes en regardant les outils utilisés et le parcours de l'APT. Si ces derniers diffèrent suffisamment d'un groupe à l'autre, il devrait être possible de construire un modèle par groupe d'attaquant.

De manière général, le travail autour de la détection des APTs ne fait que commencer. Il y a deux volets principaux sur lequel se concentrer. Le premier est la détection d'attaque inconnues. De nombreuses attaques sont créées pour éviter la détection par signature, et si nous voulons être capable de détecter ces attaques, il faut donc se baser sur la détection d'anomalie. Le problème de la détection d'anomalie est le taux de faux positifs, qui a tendance à être élevé. Il faut donc élaborer de nouveaux modèles plus précis. Les récentes avancées de l'apprentissage machine ouvrent de nouvelles pistes, mais ce ne sont pas les seules pistes à explorer. Le deuxième volet est le cœur de nos contributions : la reconstruction des scénarios. Ceci est primordial pour la détection des APTs. Cela permet non seulement de remettre les attaques dans leur contexte, ce qui permet de mieux évaluer leur impact, mais aussi peut être utilisé avantageusement pour combler les lacunes dans la détection des attaques individuelles.

# Contents

# Chapter 1

# Introduction

---

*In this chapter, we introduce Advanced Persistent Threats, a new kind of attack campaign performed by skilled actors. We then present Akheros, the company where this thesis is taking place, and its solution for detecting these new threats: a fully autonomous Intrusion Detection System based on machine learning models. The work is split in three thesis including this one. Lastly, we detail our contributions to this endeavour: we link already detected and related attacks together to form potential attack campaigns and then rank those potential attack campaigns from most probably an APT to least probably an APT.*

---

## 1.1 Security issues

Information security is an always present concern. Since information systems are constantly evolving, attacks are also changing and the defense must adapt to the new information systems to be protected as well as to the new attacks they must be protected from. Vulnerabilities can stem from a host of reasons, from programming errors, to configuration errors to inherent language or protocol flaws. As such, it is reasonable to think that any system is vulnerable all the time, which means that it is necessary to monitor those systems for signs of attacks. That way, when an attack is found, it can first be stopped then investigated to find and remove the vulnerability which enabled the attack.

These security monitoring applications are called Intrusion Detection Systems (IDS), and there are two main categories. The first kind is based on misuse detection. The idea is that the IDS knows a list of attacks and will look for telltale signs, raising an alert when it finds one. This is typically what an antivirus does: it knows a list of viruses

and looks for them in the files present on the system. This kind of IDS works very well for already known attacks. However, it suffers from a number of shortcomings. The first and biggest one is that it cannot detect unknown attacks. If an attack is not in its database of known attacks, the IDS will never detect it; these attacks that are not detected are called false negatives: negatives because the IDS does not detect an attack, false because the IDS should detect an attack. And this works for modifications of known attacks, though, in this case, the database of known attacks can be made to match attacks that are close to known attacks. However, by trying to detect variations of known attacks, the IDS increases the chance of having false positives, i.e. time when the IDS will raise an alert while there are no attacks. How much of a deviation from known attacks an IDS can detect becomes a trade-off between false positives and false negatives. If we want to detect larger deviations, we increase the risk of false positives while decreasing the risk of false negatives. Note, though, that neither false positives nor false negatives can ever truly be eliminated.

The second kind of IDS are based on anomaly detection. In contrast with the first kind, this second kind of IDS knows the system to protect and will raise an alert whenever the system performs actions it is not supposed to, hence the name anomaly. This means that these IDS do not need to know anything about the attacks to detect them. However, they do need a model of the system to be protected, and that model must be updated whenever the system changes. This second kind of IDS does not escape the false positive and false negative issue. The model of the protected system cannot fully encompass every single aspect of the system. There are times, during normal operation, when the system will deviate from the model. The IDS must allow some deviation from the model to avoid raising too many false positives, but at the same time, allowing deviations increases the risk that attacks will not be detected, i.e. false negatives. Once again, there is a trade-off to be made between the amount of false positives and false negatives that the IDS will have.

False negatives are bad for obvious reasons. If an attack is not detected, then it will not be reacted to. However, false positives are bad too. Alerts raised by an IDS are acted upon by humans. If there are too many false positives, the human operators will experience alert fatigue and will start ignoring alerts from the IDS. Every single false positive takes time to investigate. If there are too many false positives, then there may well be too many alerts and not enough time to investigate them. Either alerts will be ignored because there is no other choice or more humans operators will be needed. This phenomenon is compounded by the fact that attacks are actually rare and even with a low rate of false positives, there could well be more false positives than true positives, i.e. actual attacks being detected. Imagine a system where the IDS analyses 1000 events

per hour. If the false positive rate is of 0.1% and there is one attack per day, this means that in one ten-hour day, the IDS will raise 10 false positive alerts and 1 true positive alert. In such a situation, the human operators would quickly dismiss every alert from the IDS without even analysing it.

As we can see, an IDS should be able to detect known and unknown attacks and must have low enough false positives and false negatives to be useful. The attacks, however, change every day, with new attack vectors being leveraged all the time. This is a difficult proposition for the IDS. In addition, a recent development is the apparition of groups of attackers who are not only highly skilled but working for rich and powerful entities, such as governments and large corporations. These groups of attackers have skill, time and money. With their backing, their attacks are usually motivated by political or economic espionage and sabotage. These new kind of attack campaigns, dubbed Advanced Persistent Threats (APTs), leave the targeted defenders with little to defend themselves with.

## 1.2 Presentation of Advanced Persistent Threats

The main characteristics of an APT is that it is being executed by a group of skilled attackers backed by a powerful entity, and the attack is a targeted one. Other than that, it is mostly a normal attack campaign. The attackers will first learn as much as possible about the system they want to attack. They will find a point of entry and, once inside, they will explore the system in more details to find their original target. This will usually entail finding other vulnerable hosts to pivot to so that, little by little, they can get closer to their goal. Finally, they will reach their goal which can consist of exfiltrating sensitive data, once or regularly, or interfering with the core process of the system. The basic phases of the attack campaign are the standard phases of any attack campaign.

What makes these campaigns different is really the attackers themselves. Firstly, they have very precise goals, decided by the backing entity. This makes the campaign targeted, and means the attackers will do whatever they can to reach their goal. In contrast, during opportunistic attacks, the attacker will look for a number of specific vulnerabilities on a large number of system and exploit those that are vulnerable. By keeping software up to date, i.e. by patching vulnerabilities as they are discovered, the defender can stave off the majority of opportunistic attacks. On top of which, having robust in-depth defense system should catch most of the remaining opportunistic attacks. In the case of APTs, the attackers are capable of finding new vulnerabilities, called 0-day, and exploit them, thus patching software, while always a good practice, will not be enough to deter them. In addition, if the attackers find an already known vulnerability, they can customise an

exploit so that the IDS protecting the system will not detect it. This makes it really difficult to protect against APTs using the usual misuse-based and anomaly-based IDS.

A recent example of an APT campaign is the Monsoon campaign, analysed by Forcepoint in [Forcepoint 2016]. According to the report, the aim of the campaign is espionage. The attackers targeted a number of different industries in China and government agencies in southern Asia. There are no certainties concerning the initial reconnaissance phase, but it seems likely that the attackers either had a list of targets or a list of themes and regions and established the list of targets themselves. Then, for each target, they would send a customised phishing email containing news concerning topics of interests to the victim. This phishing email also contained a weaponised document and enticed the victim to open it. By opening the document, the victim would activate the exploits and the attackers would gain access to the victim's computer. This access would then be used to find and extract sensitive documents. Forcepoint identified two malware developed by the attackers, BADNEWS and TINYTYPHON; these tools used non-traditional command and control channels, such as GitHub, forums and news feeds. The attackers also used already known tools, such as Metasploit, an open source tool used for penetration testing, and available malware. Overall the campaign reached over 110 different countries and 6300 IP addresses; the exact count of victim is difficult to establish since a single victim can have multiple IP addresses over the course of the attack and can travel to different countries.

These targeted attack campaigns, operated by skilled attackers backed by powerful entities, are indeed a serious threat. And IDS solutions are evolving in order to be able to stop them. Since the attackers are capable of developing their own exploit specifically made to evade detection solution, these IDS must stay ahead of the attackers. For now, there are fledging solutions, but more work is required. This thesis is part of a project whose aim is to develop such an IDS. The project is separated in three parts, and we concentrate on the third part: the aim of this thesis is to identify APTs from recorded attacks inside the monitored system. While there have been work on detecting multi-step attacks, what differentiate APTs from these attacks is that APTs are composed of multiple complex attacks. These attacks can target different part of the system and can last for several months or even years. Multi-step attack detection methods, often based on temporal proximity of alerts, cannot be used in such cases, where not only there can be significant delay between each attack, but the attacks may not even happen on the same machines inside the system. In addition, APTs do not have a set path where one attack is always followed by the same next attack. Instead, the attackers adapt to the circumstances, and APTs can branch out, with the attackers usually setting several bases of operation, some used as backups in case of detection, and from there can execute

several lines of attacks in order to fulfill their goals. This is why tools tailored for the detection of APTs are necessary.

## 1.3 Our solution: Akheros

Faced with these new attacks and the lack of appropriate defenses, Akheros decided to create a fully autonomous IDS specifically geared toward detecting unknown threats in an enterprise network. Akheros is a small company fully dedicated to the creation of this IDS. Akheros' founder, Philippe Baumard, had the experience of using machine learning, and specifically Bayesian networks, to model the behaviours of humans. He hoped to use the same approach to model the behaviours of information systems, and create an IDS which could use these models and he patented the idea in [Baumard 2015]. This IDS would be inspired by anomaly-based IDS, but would go further by having not a single static behaviour but numerous behaviours with a governing behaviour for the dynamics of changes from one behaviour to the next. He called his approach incongruity detection because, with those behaviours, they would be capable of predicting the future behaviour of the protected system and compare the predictions with the measured reality. Those predictions can include behaviour changes and a failure to change behaviour would be just as severe as an unexpected change. In addition, the IDS would also model the behaviour of the system during attacks so that, once identified, those attacks could be re-identified if they reappeared at a later date. Based on this intuition, he founded Akheros.

The project is organised around three key points, each being solved by a PhD student in a separate module. The PhD students were hired specifically to work on these modules. In addition, they collaborate on creating the supporting architecture collecting the data and communicating between the three modules. The first module learns the behaviours of the system and detects incongruity, the second module assesses the potential security risks of those incongruities and the third module tries to establish links between attacks in order to find APTs and other attack campaign. The architecture of those modules as well as the supporting probes is shown in Figure 1.1 on the next page. As mentioned, the aim of the project is to produce an IDS protecting enterprise networks. Such networks are composed of multiple heterogeneous machines, some used as workstations with a dedicated users and others used as servers for either inside or outside facing services. The aim is for the IDS to collect data from all of these heterogeneous sources in order to have a global view of the network to protect.

The first module is the work of Matthieu Hourbracq. It is tasked with learning a dynamic model of the behaviours of the monitored system. It is detailed in two ar-

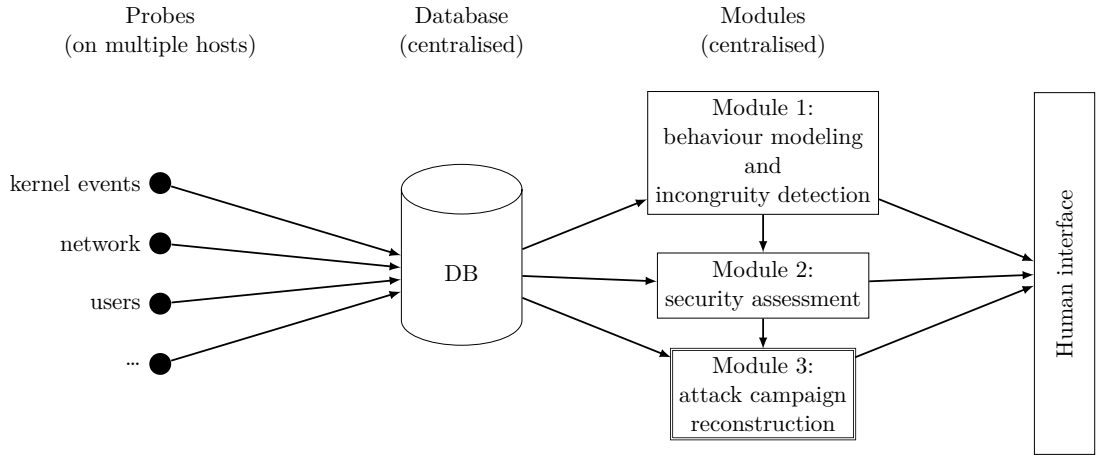



Figure 1.1: Architecture of the Akheros IDS.

ticles [Hourbracq et al. 2016; Hourbracq et al. 2017]. The original idea is to learn a Bayesian model of the system in a completely unsupervised and continous way. This Bayesian model of the system can then be used to make predictions about the future behaviour of the system. Since the behaviour of a system changes over time, this module has to take that fact into account. This means that, as it learns a behaviour, it can identify when a break occurs and the current behaviour is no longer a continuation of the previous behaviour but an entirely new behaviour instead. Additionally, the module learns the behaviour dynamics i.e. the meta behaviour dictating when the system adopts this or that behaviour. The module then uses these models to check if the behaviour of the system is consistent with the behaviours learnt until now. To do so, the module checks if the current behaviour could have been extrapolated from the known models and the meta-model. If it could not have been, an incongruity score is computed. This incongruity score takes into account how much of the current behaviour could be explained by existing models and by how many. It also checks, through the meta model, if a new behaviour is expected; this means that unexpected changes in behaviour or expected but not observed changes also contribute to the incongruity score. In effect, an incongruous event is one that is not consistent with what we know of the system and thus cannot be explained and could not have been predicted. In addition, the module continues to model incongruous behaviour so that if they are later identified as malicious they may be instantly recognised the next time they are observed. This is the first level of alerts in Akheros: incongruity alerts.

Once an incongruity alert is raised, the second module is in charge of deciding whether the alert is related to a security risk. This is the brainchild of Mark Angoustures, and it is detailed in [Angoustures, Erra, and Di Bernardino 2017]. In order to assess

an alert, the module creates a dependency behaviour graph of the system at the time the alert is triggered. This graph is made of the process tree and their actions in a sliding time window. Actions are classified beforehand in several categories by using the same dependency behaviour graph for known malware and known safe software. Each action has a category and a score in that category. If an action is only ever observed in malware, it will have a higher score than if it is observed in both malware and safe software. Then, per category, a personalised pagerank is computed on the dependency behaviour graph. This personalised pagerank highlights small clusters of processes which are responsible for the malicious actions. The idea is that if a malicious process spawns a process to do a malicious action on its behalf and then spawns a second process to do a second malicious action, the personalised pagerank is able to highlight the original malicious process instead of stopping at the process directly executing the malicious action. Finally, depending on several criteria, such as the number of malicious actions and the number of categories of malicious actions a single cluster is responsible for, the module decides whether the incongruity alert is, in fact, a security alert. In the event where an incongruity alert is analysed as presenting a security risk, the fact that the module scores the risk over several categories helps analysts confirm the results. In addition, whatever the result of the analysis, they can be integrated into the list of known malware or safe software, which will help when analysing future alerts. This is the second level of alerts in Akheros: security alerts.

The third module is the subject of this thesis. Once incongruity alerts have been analysed and found to present a security risk, this module will check past alerts to see if some could be related to this latest alert. The aim is to reconstruct attack campaign and hence highlight APTs. In addition, this gives more data to analysts so that they can respond to the threat more appropriately. For example, when we detect an attack which seems to have a low impact, if this module can link it with other attacks and show that it is part of an attack campaign, then the actual impact of this attack is now much higher and defenders can respond with this information in mind. This module operates using a two step approach. First, it tracks information flows between processes to find potential links between attacks. Then it uses a Hidden Markov Model (HMM) of attack campaign to check if the potential chain of attack it found is consistent with other attack campaigns. This is the third level of alerts in Akheros: APT alerts.

## 1.4 Our contributions

The contributions of this thesis are threefold. The first contribution establishes links between attacks in order to find potential attack campaign. This is based on the use of

Information Flow Tracking (IFT) and is detailed in [Brogi and Viet Triem Tong 2016]. The second contribution uses a Hidden Markov Model (HMM) to model attack campaign. This is used to score the potential attack campaign found by the first contribution. The approach is detailed in [Brogi and Di Bernardino 2018]. In particular, it introduces a score which is robust to missing observations, i.e. attacks that were not detected. The third contribution is the introduction of a tool we created to define, replay and share complex attack scenarios called Moirai. This was also the subject of an article [Brogi and Viet Triem Tong 2017] and is detailed in Section 5.4 on page 85.

In order to perform IFT, the module uses the raw data from the database with additional indications from the first two modules showing which events in the data raised alerts and which alerts were seen as security threats. Whenever a new alert is raised, the module associates a new tag with it and starts propagating it. The propagation is done by transforming the events collected into flows of information between containers of information. In this case, those are processes, files and sockets. These containers are progressively tagged when they are the receiving end of an information flow which is either part of an attack or whose originating end was already tagged. Container by container, the tags slowly propagate and a link between attacks is established whenever a tag is propagated by an information flow which is also part of an attack. A simple example would be an attacker exploiting a vulnerability in a web application to upload and execute a program. This attack is detected, a new tag is created and one of the flows of information part of the attack is the one where the web application writes the program to the disk. The file containing the program, which is a container, is thus tagged with the tag of the attack. When the program is executed, this second flows means that the process running the program is also tagged with the tag of the attack. When this process is used to launch a second attack which is also detected, the information flows part of this second attack originate from the process. Since the process is tagged, the module knows there is a link between the two attacks. Things work well in this simple example and as long as we do not miss flows of information, IFT will be able to link related attacks together. However, IFT will also link unrelated attacks together, especially since we track information at the scale of whole files and processes. In a second example, a first attack occurs on the web server, apache. A new tag is created for the attack and this tag ends up on the apache process. When a second attack uses a vulnerability in apache again, there will be flows of information part of this attack which originate from the apache process. Since the apache process is tagged, a link between the two attacks will be established, even though the two attacks are not part of the same campaign. This is one of the inherent drawback of IFT. We add tags to the system, but there are no mechanisms to remove them, and so, with time, more and more links will

be established between unrelated attacks. The method is fully detailed in Chapter 3 on page 39.

This is where the second step of the module comes in. The aim of this second step is to filter real links from spurious links. We use a HMM for this. A HMM is a stochastic model with the following properties; firstly, the process we want to model follows the Markov hypothesis meaning that the state of the process at a given time only depends on the state of the model at the previous time; secondly, the process cannot be observed directly but its progress can be inferred through indirect measurements. This matches what we want to do. Attack campaign can be split into phases, these phases are the state of the unobservable process. It is unobservable because we do not know in which phase the attacker is. However, the actions of the attacker will change depending on the phase of the attack, and so we can use those actions, which are observable and lead to alerts, to infer the current phase. Note that there is some overlap between the actions and so we cannot know with absolute certainty the phase based on the actions. In addition, the current phase of an attack campaign depends on the previous phase and whether it succeeded or not. If it succeeded, the attacker is very likely to go to the next phase and if it did not, they are very likely to try another angle of attack, either in the same phase or in an earlier phase. So, as we can see, an HMM seem to fit with the requirements for this second step. In addition, by using an HMM, we can not only check out whether a chain of observations fit the model, but we can use the Viterbi algorithm to infer the most likely chain of states to have generated the original chain of observations. This means that we can associate each attack in the chain with a probable phase of the attack campaign, which means even more information for the analysts: not only do they know that the attack is part of an attack campaign, they know which phase of the attack campaign each attack is in. This is the subject of Chapter 4 on page 55.

There is one point that we have not addressed yet. Since APTs are performed by skilled attackers, there is chance that some of the attacks will not be detected by the first two modules. Both the IFT step and the HMM step must be robust to these undetected attacks. IFT is actually inherently robust to undetected attacks because attacks do not change the way tags are propagated, they only introduce new tags. IFT finds link between attacks when a flow is part of an attack and the origins of the flows are tagged with the tag of another attack. The end result of propagating this flow is that the containers where the flow ends are tagged with the tag of the new attack. If that new attack is not detected, these containers are tagged with the tag of the old attack. The tag on the containers is different, but there is still a flag. It will be propagated normally and if it is on the origin of a flow which is part of a third attack, then that third attack will become linked to the previous attack. If the second attack was detected, we end up with the

first attack linked to the second attack which is linked to the third attack. If the second attack is not detected, we end up with the first attack linked to the third attack; which is the best we can hope for since the second attack was not detected. This means we only need to work on the HMM to make sure that it takes into consideration possible missing steps. To do this, we modify the Viterbi algorithm, used to find the most probable chain of states based on the chain of observation. We add the possibility that between two observations, the chain of states has state without observations, i.e. attacks that were not detected. When computing the score, we can choose the probability that an attack will be detected. We use this probability to weigh the possibility of missing observations in our modified Viterbi algorithm. This customised Viterbi algorithm then gives us the most probable states which generated the observations with its score, taking into account possible missing observations. With this solution, both of our steps are resilient to the possibility of undetected attacks.

The rest of this thesis is organised as follows. We will present related work on APT detection, Information Flow Tracking, Hidden Markov Models and data generation in Chapter 2 on the next page. We will then detail our first contribution, linking attack using Information Flow Tracking in Chapter 3 on page 39. This includes an evaluation. We move on to the Hidden Markov Model and the modified Viterbi algorithm, including an evaluation, in Chapter 4 on page 55. In Chapter 5 on page 77, we present the tools used in the evaluations in detail. In particular, we explain how we obtain reproducible scenarios using Moirai. Finally, we will conclude this thesis in Chapter 6 on page 89.

# Chapter 2

# State of the art

---

*In this chapter we discuss related work. We split the chapter in four parts. First, we focus on other approaches to detecting APTs since this is the global goal of this thesis. We use these related work to confirm the generic phases of an APT. Second, we present the state of the art of Information Flow Tracking. As we will see, IFT is usually employed to find single attacks, but the implementations are still applicable for our specific application. Third, we do the same of Hidden Markov Models. We detail their different uses and the usual scores and their specificities. While these scores are not directly applicable to our use case, we still keep their general semantics. And, fourth, we present the possibilities available for evaluating Intrusion Detection Systems. This is important because we need to evaluate our solution and, as we will see, the available datasets are heavily criticised and not made for the evaluation of APT detection in particular.*

---

## 2.1 APT detection

### 2.1.1 Definition and examples of APTs

In recent years, Advanced Persistent Threats (APT) have emerged as a real and possibly devastating security risk for companies and governments alike. An APT is a targeted campaign of attacks, usually executed for political or economic reasons [Websense 2011]. These campaigns are executed by talented and determined attackers, sometimes sponsored by nation-states. In contrary to opportunistic attacks, this implies that attackers will spend the time necessary to find a weak point and, once inside, will lay low as long as needed in order to achieve their goals. Overall APTs can take months or even years to

unfold, especially if one of the goals is continuous data exfiltration. As such, the combination of targeted attacks executed by competent attackers with time and resources on their side makes APTs real and serious threats.

APTs may be customised to their target, their evolution usually follows the same broad pattern, as described in [Sood and Enbody 2013; Tankard 2011]. After a first phase of reconnaissance to find a weak point, the attackers perform an initial compromise in order to penetrate the perimeter. Since the weak point is not necessarily on the host where their goals lay, the attackers will establish a foothold and try to elevate their privileges while also performing internal reconnaissance to find weak points inside the perimeter which could bring them closer to their goal. Thus, they repeat the "find weak point", "exploit weak point", "establish foothold" and "elevate privileges" loop until they can get to their goals. Once they finally have a foothold, possibly with elevated privileges, on a system from which they can access their goal, the last phase begins: "data exfiltration". If this goal is only one of several required one, the previous steps will loop back to the internal "find weak point". For example, if the attackers want to steal the blueprints in division A of a business, they may target an employee in division B with a spear phishing email. Once they are in, they can scan the internal network to try and find vulnerable services used by both division A and division B. Once they are inside one of those services, they will try to elevate their services and also infect a few more services in case their first site is discovered and cleaned. They will then find a way onto division A's file server where the blueprints reside and then exfiltrate them.

In [Mandiant Intelligence Center 2013; Fireeye Labs 2015; McAfee Labs 2013; Trend Micro 2013; Tankard 2011], five different teams present a post-mortem analysis of one APT family each. Each family is attributed to a group of attackers and is characterised by the set of tools used. This toolbox is reused, updated and expanded from one APT to the next operated by the same group, year after year; and they end up used on wildly different targets. For example, in [Mandiant Intelligence Center 2013], the toolbox is used over 7 years and touches at least 141 organisations in 15 countries working in every sector, from IT, of course, to the financial sector to agriculture to healthcare and education. The longest attack campaign lasted almost 5 years, during which they managed to hide, monitor their target and retrieve sensitive files. Over the years, the toolbox grows, acquiring new malware families according to the needs of the attackers. The attackers are organised like any company, with even reports of attackers working in shifts in [Fireeye Labs 2015] in order to be active at all times. In that article, the attackers' group has been active for more than 10 years. They have professional developers working on the toolbox, and the tools are versioned and update automatically. When required, they create variants specific to one of their target. Their main aim is data theft, and they have

special tools in order to retrieve data from air-gapped networked, which, by definition, should be impossible.

Overall, these analyses show, in concrete terms, that APTs are indeed a real and dangerous threat. They also show that all is not lost when trying to defend against them. The attackers are as much creatures of habit as any other humans. They reuse their tools from one attack campaign to the next, making signature based detection tools not completely useless providing one of their campaign has been detected and analysed. Even if they are skilled, they also used basic exploits if these have not been corrected. And even between attacker groups, the same behaviours can be observed. They use the same kinds of softwares in the same kinds of ways. Every single campaign in [Mandiant Intelligence Center 2013; Fireeye Labs 2015; McAfee Labs 2013; Trend Micro 2013; Tankard 2011] goes through the generic phases presented in Chapter 1 on page 1. Facing these advanced threats, advanced detection techniques could make use of the similarities present within all APT campaigns to detect them.

Additionally, even when not being able to detect APTs, there are still strategies to minimise their impact. Bowers et al. present in [Bowers et al. 2012] a model of APTs as a two player game: the attacker and the defender. Both player compete for control of a system. At any time, each player can, for a cost, take complete and silent control of the system. The aim for both players is to be in control for as long as possible while paying as little as possible. This model of a game with stealthy takeovers is used to expose the costs of being the target of an APT as well as to explore strategies to minimise it. They apply this to password reset policies, for example, and show that resetting passwords at random intervals has benefits over resetting them at fixed intervals.

In this section, we learnt that APTs pose very high risks to enterprise networks. They are operated by very skilled and organised attackers. This makes them very difficult to detect, and as proof, some campaigns have gone on for years without being detected. However, there are similarities and patterns that appear in every APT, even if when they are executed by different entities. This opens avenues to explore in order to defend against them. Additionally, even in the case where an APT is going on but has not been detected, there are strategies to adopt in order to reduce its impact. All in all, even though it is more difficult than for opportunistic attacks, fighting against APTs is certainly doable.

### 2.1.2 State of APT detection

#### Multi-step alert correlation

We must start this state of the art by a small detour to alert correlation. While our approach is different in its execution, the idea and reasons for the approach are mostly the same as the ones behind alert correlation.

In essence, alert correlation is the process by which several alerts from one or more IDS can be aggregated in a single alert. A simple example would be an aggregated alert indicating a port scan from a smattering of alerts from individual ports. The effect of this aggregation is twofold. Firstly, the human operator acting on alerts is shown less alerts; in the previous example, this means a single alert instead of a smattering of unexpected activity alerts, one for each port. Second, the alerts shown have more information since all the information from all the original alerts is aggregated in a single alert. Staying with the example, the aggregated alert can indicate that there is a port scan while the individual alert only indicate an unusual activity for a given port. This means that the operator have less alerts to respond to, and the alerts they are responding to are easier to deal with since there is more context.

While the original concept is to aggregate all the alerts from a single attack and output a single alert about that attack, it has since been extended to the detection of attack campaigns with the use of multi-step correlation. And while the concept of alert correlation predates the concept of APT, an APT is definitely a multi-step attack, also known as an attack campaign, albeit one that can take several months to unfold. Hence this small digression.

[Valeur 2006] presents a nice summary of alert correlation methods and builds on these methods to create a multi-component real-time alert correlation system.Valeur starts by presenting the three main types of alert correlation, multi-step, fusion-based and filter-based, and proposes an integrated correlation process using several correlation steps and other processing. Each type of correlation is put to use where it is most effective in order to reduce the load on each component and achieve real-time processing. Valeur then gives a basic example involving an host monitored by a host-based IDS, an application-based IDS and two network-based IDS. This host is the target of an attack campaign, called a multi-step attack in this case, and another single attack. The aim of the correlation system is to identify the multi-step attack, isolate it from the other attack, and present it as a single alert with all the associated information. This is exactly the results that we wish to achieve with this thesis.

However, the means are different. Alert correlation requires rules in order to aggregate

alerts and we want to avoid specifying explicit rules. It is possible to generate correlation rules automatically. An example is [Godefroy et al. 2014] where Godefroy et al. present a way to generate correlation rules automatically. The rules are generated from an attack tree, and since the tree is built with the knowledge of the system to protect, this gives rules that are specific to that system. However, this approach still requires expert knowledge to build the attack tree, and while the work of the expert is reduced, this is still a step we want to avoid.

All in all, our approach is similar to correlation in its motivation and expected results. However, the methods employed differ and the major reason for these differences is the fact that we want to remove expert knowledge as a pre-requisite of the IDS.

#### Explicit attack campaign detection

There already exist commercial tools which are sold as being able to detect APTs. In [Ács-Kurucz et al. 2014], Ács-Kurucz et al. evaluate the reliability of such commercial tools at the end of 2014. In order to represent the unknown and complex malware usually present in APTs, they develop custom samples. These samples implement usual Remote Access Tool (RAT) functionality such as code execution, download, upload and so on. They are then submitted to five commercial APT detection solutions. Of the four samples, only the two simplest ones are detected by all five solutions; even then, some solutions rate them at their lowest threat level. The third sample is detected by two of the five solutions and the last one is detected by none. This shows that existing solutions are easily circumvented and validates the need for better solutions. Additionally, the existing solutions seem to concentrate on detecting the steps of an APT individually. We believe that the fact that each attack in an APT is part of a campaign of attack can and should be leveraged when detecting them. This is why we focus on detecting whole or partial attack campaigns rather than individual attacks.

To compound this, a 2010 Verizon report [Verizon 2010] shows that 86% of victims have traces in their logs and are still unaware they have been breached. Thus, it is not only a problem of APT using advanced attacks which cannot be detected. Even when they do use less advanced attacks which do leave traces, they are not always detected, and in the case that they are, their severity is underestimated. One of the advantages of our approach is that even if a low severity event occurs, if we can link it to a number of other events, then it will be analysed with the appropriate level of caution. In [Li, Lai, and DDL 2011], Li, Lai, and DDL give another example from 2011 showing that APTs are not unstoppable threats completely different from usual attacks. Instead, we see that they employ the usual methods, such as phishing, when trying to get a foothold in a

system. In addition, the payloads used can be analysed like any other kind of payload, and such analysis reveal that these payloads are not necessarily unique or special. This shows that writing tools capable of detecting APTs is definitely possible.

Commercial products are able to detect only the most basic APTs and only in the best of cases. Academic proposals try to do better. In 2013, in [Virvilis, Gritzalis, and Apostolopoulos 2013], Virvilis, Gritzalis, and Apostolopoulos do a technical analysis of reported APTs in order to find common patterns and techniques. They also identify the issues that enable these malware to evade detection and explores solutions for strengthening the defenses against such attacks. While the analysis does not focus on the different phases of an APT, it shows the functionalities of the different APTs. Once again, even though the five APTs analysed are different, the basic ideas behind them are the same. They all use command and control servers and communicate on the most used ports, such as 80 or 443. They also all use encryption and obfuscation techniques.

Based on similar observations, some articles try to detect the command and control phase and the exfiltration phase monitoring only the network. The 2011 Binde, McRee, and O'Connor article [Binde, McRee, and O'Connor 2011] bases its analysis on one use case: the Aurora APT. The key observation is that APT traffic stands out during certain phases. Once a foothold is established, it communicates with the command and control center outside of the target. This traffic is usually partially automated and should stand out against the normal traffic of the enterprise, which would normally mostly stop during off hours. Additionally, when the attackers exfiltrate the data, they usually exfiltrate a lot of data at once and this should show as a peak in network usage which can be detected. While this is far from an exhaustive way to detect APTs, it shows that however advanced the attackers may be, they cannot go around the physical limits of the hardware, and so simple solutions can also help in detecting attacks.

Vance has a similar approach in his 2014 article [Vance 2014] but focuses on the cloud. Cloud providers are just as vulnerable to APTs than any other company. However, in their case, the provider relinquishes control of the higher levels and cannot, for example, do white-box monitoring of the applications running on their infrastructure. The article proposes a solution based on monitoring network flows. Flows are available to the cloud provider and can be used even if the data is encrypted. The solution presented creates statistical models of the normal network traffic. Command and control traffic and data exfiltration traffic show as anomalies against the normal network traffic model. While this approach can detect these two phases, it does not attempt to reconstruct the evolution of the APT nor does it detect the other phases.

In 2015, in [Chandran, Hrudya, and Poornachandran 2015], Chandran, Hrudya, and Poornachandran extend the reasoning to indicators on the monitored systems like CPU

usage, RAM usage, number of files on disk, and so on. Instead of defining rules on each indicator to limit the values it can take, they use machine learning to classify the systems as clean or infected. They compare different techniques to see which ones give the best results. The exact methodology is unclear, but they do manage to have 99.98% accuracy with their best performing technique. While they do not attempt to reconstruct APTs, their experiment show that even advanced malware changes the behaviour profile of the system when they act, and thus can be detected.

Similarly and also in 2015, in [Friedberg et al. 2015], Friedberg et al. start from the observation that pre-defined rules are not enough to stop customised and targeted attacks, especially for applications with low market-share for whom such rules may not even exist. Hence, they propose applying machine learning to network event correlation in order to create an anomaly based IDS. They capture events from system logs and use unsupervised learning. Their approach extracts patterns from each line and qualifies those patterns. One line can be affected to multiple classes. Hypothesis are then made on the implications of one line on the next based on these classes. Hypothesis that hold for long enough are then turned into rules. These rules are then used to detect anomalous behaviours. They apply this method to the monitoring of Industrial Control Systems. Most of the components in these systems are automated systems so such an approach fits well. However, even if they do attempt to detect unknown attacks, they do not try to detect APTs, and, in particular, they do not broach the subject of finding separate attacks which could be part of the same APT.

Y. Wang et al. take a different approach in their 2014 article [Y. Wang et al. 2014] and introduce the concept of network gene in order to detect APTs. A network gene is defined as the digital segments extracted by network protocol reverse analysis and their combined sequences. The sequences are important because they represent the network application with semantic-rich network behaviour patterns. There are three levels of network genes from low to high: the messages, protocols and operations. Together, these three levels form the network genome of the application. The genomes of two applications can be correlated in two ways. The first case is when two higher level genes share the same lower level gene. The second case is when two genes on the same level only present some minor variations. The gene pool is created by analysing a large number of network applications already classified in benign, malignant and neutral applications. Neutral applications are applications which are unknown and cannot be classified in either benign or malignant. Once the gene pool is created, the network is monitored, and network traffic is filtered in real time according to the white and black lists in the gene pool. Once again, this concept is good to detect unknown attacks but does not really detects APTs as there is no way to find attacks which are part of the same attack campaign.

As we can see, there are ways to detect unknown attacks. However, the previous articles only detect individual attacks and do not use the fact that those attacks are part of a larger attack campaign. Articles more relevant to our approach use the links between attacks in an APT in order to reconstruct the attack campaign. In [Hutchins, Cloppert, and Amin 2011], in 2011, Hutchins, Cloppert, and Amin present an intelligence-driven intrusion kill chain. Being a more military oriented article, they recognise the following phases for intrusions: "reconnaissance", "weaponisation", "delivery", "exploitation", "installation", "command and control", and "actions on objective". Since each phase requires that the previous phase completed successfully, the aim of the kill chain is to prevent the current phase from completing in order to cut the chain. The possible actions are "detect", "deny", "disrupt", "degrade", "deceive", "destroy". Each action will take a different form depending on the phase of the intrusion, and some are not even available for a given phase. For example, it is not possible to "disrupt" a "reconnaissance" action, only to "detect" or "deny" it and "weaponisation" will be "detect"ed by a network-based IDS while "exploitation" will be "detect"ed by a host-based IDS. The phases of an APT used in this article are more precise than what we used because the precision is needed in order to give a proper response. In our case, we do not need to split "weaponisation" and "delivery" or "exploitation" and "installation". Other than that, this article gives little details on how to do the actual detection and how to decide which phase the intrusion is currently in.

In their 2014 article [Bhatt, Toshiro Yano, and Gustavsson 2014], Bhatt, Toshiro Yano, and Gustavsson also use a kill chain model to present a framework for the detection of APTs. Their idea is to use a layered defense to slow down the progress of the attackers. At the same time, this means that attackers will have to perform more actions to get to their goal and thus increases the chances that they will be detected. For each phase of the kill chain, they identify the attack vectors. For each of these vectors, they define prevention and detection measures. Some of these measures includes IDSs and other automated monitoring system, all of them off-the-shelf components. These systems feed their data to a centralised database. The database is analysed by an intelligence module which is already aware of kill chain patterns and malware patterns. The patterns are created using a separate malware analysis module. They show that with this architecture, they are able to detect their experimental APT. Once an alert is detected, the intelligence module tries to determine which phase of the APT the alert belongs to and then looks for the evidence of the previous steps. This article is interesting because we see once again that the same simple model of APT is used. Additionally, they also use the fact that it is the chain of attacks which makes the APT and look for the previous phases when they detect an attack. Sadly, this part of the intelligence engine is not explained.

Also in 2014, Bukac, Lorenc, and Matyáš propose an extension of the kill chain, in [Bukac, Lorenc, and Matyáš 2014], where the attacker performing an APT is allowed to continue the attack after being detected, placing the system under increased surveillance. By so doing, the defender gains the opportunity to collect valuable intelligence which can then be used to defend against future attacks. They propose that after a time of passive observation, the defender would actively tamper with the attacker in order to evaluate their full capabilities, thus transforming the system into a live honeypot. The benefits are clear. The defenders learn more about the attackers, seeing their tools in action, forcing them to deploy new ones and identifying their goal. It then becomes easier to assess the full extent of the intrusion and to detect dormant hosts. There are, however, disadvantages. First of all, there may be legal requirements to immediately end and clean-up intrusions, especially if private data may be at risk. Then, there is the risk that the only part of the attack has been detected and that the attacker is able to compromise the system even further. The article then presents a number of questions to answer when deciding whether to end the intrusion or observe it further.

While De Vries et al. do not mention kill chains in their 2012 article [De Vries et al. 2012], the analysis framework they present to detect APTs is certainly related. It starts by splitting APTs in 6+2 phases. The six phases are the usual ones, and the additional two are "control of information leaks" and "erasing tracks" and are active during most or all of the other phases. For each phase, they list the attack methods used, the attack features which can be detected, the network locations of these attacks, the methods to use to detect them, the type of techniques the detection methods use and, finally, the impacted business aspects. These listed features help answer the classic "What?", "Where?", "Why?" and "How?" which are then used to select the best method to defend against APTs based on the needs of the defender. The article does not present an actual attempt at detecting APTs but instead presents ways of thinking about APTs in order to detect and defeat them. We can see that the other articles presented here conform in some way to this article. However, in order to conform to everything in this article, each potential victim would have to develop their own APT detection solution, which is not a reasonable expectation.

In [Giura and W. Wang 2012] in 2012, Giura and W. Wang first present the main stages of an APT. Based on this model, they introduce the concept of attack pyramid, an extension of attack trees. An attack tree represents a threat, and is built recursively with the goal as the root of the tree and ways to reach that goal as children, who are then treated as new subgoals. In the resulting tree, each path from a leaf to the root represents an attack path. A plane is the layer upon which an attack occurs, such as the physical plane where the attacker can pick a lock, the network plane where the

attacker compromises a server or the user plane where the attacker sends a phishing email. Typically, in an attack tree, a path will never cross planes. Since an APT will usually operate over a number of different planes the authors have introduced the concept of attack pyramid. An attack pyramid is an attack tree where paths can cross planes. Thus, with each plane representing a different attack surface, they correlate events across planes and can show attacks unfolding on several planes at once. Correlation is done through a set of rules. These rules are defined for each combination of two planes on common attributes. Correlation only shows the link between events, and so a second set of rules is used to detect security alerts. They use three types of rules: signature based rules which compare observations and known attacks and which must be constantly updated; profiling based rules which compare observed profile and behaviour with profile and behaviours baselines which also need to be updated whenever the baseline changes; and policy based rules which are based on organisation policies and must be updated whenever these policies change. Finally, whenever these rules are triggered, a confidence factor and a risk factor are computed from the confidence and risk factors of the individual events. If the confidence and risk factors rise above given thresholds, then an APT alarm is triggered. This work is interesting because it specifically addresses the detection of APTs and presents attack pyramids to represent them, which takes into account the fact that an APT can operate on several planes. However, it requires that the potential goals of an APT be identified, and for each goal, every path leading to this goal must also be identified; and they must also be updated whenever the system changes. This is specifically something that our approach wants to avoid, because it involves a lot of work to set up and maintain properly.

In [Sexton, Storlie, and Neil 2015], Sexton, Storlie, and Neil decompose an APT into five phases, “delivery”, “exploit”, “install”, “command and control (C&C)”, “actions”, which they call the attack chain. Contrary to the attack pyramid which starts from the goal and reconstructs possible attack paths, the attack chain is a model for APTs. This is more flexible in that there is no specified way to go from one phase to the next. Instead, each phase is associated with a number of event types indicative of this phase; events in different phases must then be combined before an APT can be detected. These events are not necessarily, by themselves, indicative of a security breach. The combination is done as follows. First, each event is assigned a score based on historical data, either on the host it occurred on, or on all the hosts; the more an event is unexpected, the higher its score. p-values are then computed for each event type and for each host; next, p-values are computed for each attack phase for each hosts, which is done by retaining only the smallest p-value of these event types, i.e. the least expected event type; these p-values are then multiplied and the anomaly score for a given host is the negative logarithm of

this number. An anomaly score is also computed for clusters of hosts where the same event types occur, this is done in a similar fashion, and enables highlighting clusters of potentially anomalous hosts. While this approach starts from the same model of APTs as the article from Giura and W. Wang [Giura and W. Wang 2012], the method for obtaining the likelihood a cluster of hosts is the target of an APT differs significantly. However, this approach is based on anomaly detection and as such requires a significant investment in understanding the monitored system and the attackers. In particular, each event that can be observed must be attributed to a phase of the APT ahead of time.

Much earlier, in 2004, in [Gladyshev and Patel 2004], Gladyshev and Patel were already proposing to model systems as finite state machines in order to conduct forensics analysis. Once a model of the system has been created and the current state identified, the model can be used to reconstruct the path leading to the current state. While this model does not identifies attacks, it does highlights the possible sequences of events leading to the current state, and which states have to be transitioned through. Some states could represent attacks, and this method could then reconstruct attack sequences. In a way, this is similar to what we do since we also identify the sequence of events in an attack campaign, our approach does not require a model of the system.

In order to counter anti-forensics measures, C. Liu, Singhal, and Wijesekera use attack graphs and augments them with anti-forensics steps to explain missing links in their 2012 article [C. Liu, Singhal, and Wijesekera 2012]. The aim of attack graphs is to reconstruct the attack path used by the attackers. However, attacker can use anti-forensics tools to erase some of their traces and thus break the links between attacks. By adding the anti-forensics node, the paths can be recovered and the attack campaign properly detected. This is interesting because, not only are they trying to reconstruct attack campaigns, they also take into account non-detected attacks which would otherwise break the links between the rest of the attacks. While missing attacks do not break the links in our approach, we do have to take them into account when evaluating potential attack campaigns identified by IFT.

In summary, from these articles, we can take that in order to detect APTs, we must be able to link related attacks into attack campaigns and only then make a decision whether this attack campaign is indeed an APT. We do have the advantage that we do not need to detect individual attacks themselves, as this is done in the previous steps of the Akheros IDS. Thus, we are left with the sole task of reconstructing attack campaigns and deciding on their representativeness of an APT. For the reconstruction of attack campaigns, we do want to use the simple model for APTs present in the above articles: “reconnaissance”, “compromise”, “establish presence”, “privilege escalation” and “mission completion”. However, we do not want the model to be too strict, neither

in the chaining of the steps nor in the captured events representative of each steps. Hence, a probabilistic model seems appropriate and more specifically, a Hidden Markov Model has all the required characteristics. The phases of an APT cannot be measured directly, but must instead be guessed at depending on the types of attacks seen; an APT starting with three "compromise" steps, while less likely, will still fit the model; and, while there will be more likely types of attacks for each step, each step can still be represented by less likely types of attacks. However, such a model cannot separate interlaced attack campaigns. Instead, our approach must be divided in two steps. The first one will use Information Flow Tracking to find links between attacks. This should reconstruct attack campaigns, while separating unrelated attacks occurring at the same time. The intuition is that, for related attacks, there should be flows of information from one attack to the next as the attacker uses the tools setup in one attack to perform the next one. Unrelated attacks should not have flows of information between them, and so we should be able to clearly separate different attack campaigns. However, Information Flow Tracking does tend to have a lot of false positives. Thus, the second step will evaluate whether the reconstructed attack campaigns are indeed APTs using a Hidden Markov Model. The Hidden Markov Model can capture the flexibility of attack campaigns so that we should only need one generic model to recognise all, or most of, the attack campaigns.

## 2.2 Information Flow Tracking

Information Flow Tracking (IFT), also called tainting, is a method used to follow the propagation of information inside a system. Implementing Information Flow Tracking requires three things. The first thing are the containers. These are the objects that contain information and they are the origins and destinations of flows of information. The second thing is the flows of information themselves. These indicates that information is being sent from one container to another. The last thing is taints, hence the name tainting. Taints are metadata that can be attached to containers to indicate the data that has flown through them. The exact rules defining how taints are attached to containers vary depending on the IFT scheme and its purpose. In our case, we want to use IFT to track the flows of information starting at each attack and see if they end up at another attack.

IFT can be implemented at different levels and in different ways depending on the goals of the approach. Suh et al. present a dynamic hardware implementation with a custom CPU in their 2004 article [Suh et al. 2004]. Its aim is to protect programs against malicious software by identifying spurious inputs and restricting their usage. The approach is based on the observation that attacks which take over other programs must,

at some point, transfer control to malevolent code. The aim is, thus, to prevent spurious data from being used as a program's control. In order to do this, every potentially malicious input channel must first be identified. When the system runs, a module in the OS marks data from these channels. Then, during execution, the CPU determines whether the result of an instruction is spurious based on the inputs and the instruction. The CPU prevents spurious data being used in dangerous ways, such as being the target of a jump. This approach, thus, prevents the exploitation of buffer and stack overflows (but not the overflows themselves) without modifying the programs or knowing how each overflow works. This is an interesting approach to show th capabilities of IFT. However, it is not practical as it requires implementing a custom CPU.

To add more flexibility, Dalton, Kannan, and Kozyrakis present, in 2007, in [Dalton, Kannan, and Kozyrakis 2007], a hardware and software hybrid IFT platform. The aim is to block high-level semantic vulnerabilities such as SQL injections and cross-site scripting, which [Suh et al. 2004] does not stop, while also detecting lower level vulnerabilities like buffer overflows. To achieve this, Raksha, the proposed architecture, relies on a custom CPU as well as a software component. This hybrid approach is meant to shore up the weaknesses of the individual approaches. Approaches based on custom CPUs are very precise in their Information Flow Tracking and do not require the programs to be modified, however the policies they enforce are not flexible at all. At the other end of the spectrum, software-based approaches have flexible policies, but their tracking is not quite accurate and they require that either the source or the binary of the program be modified. In Raksha, the CPU is tasked with tracking the information flows, thus the tracking is precise and the programs do not need to be modified. The policy enforcement is done in software, which means it is flexible and can be changed at runtime. It is still limited by the hardware, and in Raksha, the limit manifests as the maximum number of active policies being limited by the hardware. This approach melds the precision of hardware tracking with the flexibility of making decision in software. However, once again, it relies on a custom CPU which makes it difficult to deploy in the real world.

In [L. C. Lam and Chiueh 2006], in 2006, L. C. Lam and Chiueh present a software based IFT framework, GIFT. Instead of modifying binaries, GIFT is a compiler for the C language, and in order to use it, programmers must modify their code to include a function which adds tags whenever data enters the program, a function which sets the tags whenever a variable is assigned a value, and a function which decides what to do when data exits the program; it could, for example, prevent writing data with a certain tag. The compiler then inserts these functions in the code whenever necessary, so that the actual source code of the original program does not need be modified. GIFT also provides functions to access the tags at runtime. The article demonstrates the use of these

capabilities by building an automated sandboxing framework for network applications, such that the sandboxing is only enabled if necessary. In their evaluation, this sandboxing framework adds about 35% overhead. While this approach does not require a custom CPU, it requires instead that every program be created with IFT in mind, which is an even less reasonable assumption.

Tracking can also be done by modifying binaries. An example is the 2006 article by Qin et al. [Qin et al. 2006]. The approach leverages dynamic binary instrumentation to implement IFT. The advantages of this approach is that it is software only, and so removes the need for custom hardware. Additionally, being binary instrumentation, it does not require the source code of the program, and so will work with any program, even if it was not designed for IFT or if the source is not available. The design is focused on lowering the overhead of IFT, which it does in three ways. First, it only computes information flow if the inputs are not already tainted, which is the majority of cases. When the inputs are tainted, the running code is split into logical blocks where the flows of information can be directly computed from the inputs to the output of the block, thus reducing the number of checks. Lastly, they work to reduce the number and overhead of context switches between the original program and the IFT code. With these optimisation, they manage to drastically reduce the IFT overhead compared to previous binary instrumentation solutions. A number of policies are implemented on top of this framework so that for example, much like in [Suh et al. 2004], unsafe data cannot be used as the destination of a jump. While this approach works on any compiled software, it does require that each binary on the system be modified, which is not practical.

In their 2008 article [Ruwase et al. 2008], Ruwase et al. improve the performance of dynamic IFT by parallelizing it. In order to be able to parallelize IFT, it must first be relaxed, as in limited to unary operators. The log of information flows is then cut in segments, with one segment by workers. The workers, then, compute the state at the end of their segment relative to the beginning and send the results to a master thread. The master then uses the results from all threads to check information flows for the whole program. With regard to performance, once the log is large enough, even a small number of workers can give a significant speedup with this method. Since IFT can have a large overhead, depending on the implementation, any method to speed it up is interesting. The problem here is that this method limits IFT to unary which does not fit with our use case.

Yin et al. propose, in [Yin et al. 2007] in 2007, to use system-wide IFT to detect malware, in particular malware looking for sensitive data. To this end, they present Panorama, an end-to-end solution to analyse malware samples automatically and look for malicious information access and processing behaviours. At the center of this solution

is operating-system aware hardware-level taint tracking. This affords them the fine-grained precision of hardware tracking with the ability to map the tracked objects to files and processes as seen by the user. With this solution, they are able to detect all of the malware samples in their tests while having only few false positives, even though the tested samples include a wide range of types of malware, such as keyloggers, packet sniffers, backdoors and rootkits. Their approach is interesting but limited to analysing malware offline and does not protect a running system.

In 2008, M. S. Lam et al. present PQL [M. S. Lam et al. 2008], the Program Query Language, to specify information flows patterns in order to protect web applications. The idea is to check that untrusted input is not allowed to take control of the application. The programmer can specify patterns that should be avoided, and the system does both static and dynamic analysis to check for the pattern. Additionally, using PQL, the programmer can also specify corrective actions to take whenever a matching pattern is detected. They show that with this approach, they are able to, not only, detect serious attacks in existing applications but also recover from them. This approach acts as a kind of firewall, where administrators can specify unwanted flows and how to handle them. This is definitely an interesting approach but our method is proactive and wants to avoid having to define rules beforehand.

A solution between custom hardware and modifying every program on the system is to do the tracking in the operating system. It is less precise than a custom CPU but can run on commodity hardware; at the same time, it only requires modifying one piece of software: the OS. In [Enck et al. 2014] in 2014, Enck et al. modify the Android OS in order to check that applications do not leak private data, and call it TaintDroid. They assign a different taint to each source of private data, such as GPS or contact information. They then define one data sink for each point of the system where data leaves. They leverage the fact Android applications run on the Java Virtual Machine (JVM) and cannot access this data directly but must instead use well defined interfaces. They, thus, modify the JVM to track information flows. They allow the tracking of several taints at once in order to track each source of private data independently. Additionally, the JVM uses Java Native Interfaces (JNI) in order to interact with the OS, so flows are also tracked when JNI are called. Finally they also track information flows when applications use interprocess communications and permanent storage. Their setup allows them to check whether unmodified applications do leak private data. The approach in [Hauser, Tronel, Reid, et al. 2012; Hauser, Tronel, Fidge, et al. 2013] is similar, with Hauser, Tronel, Reid, et al. modifying the Linux OS to track information flows in 2012. The first article uses this to detect confidentiality violations, just like [Enck et al. 2014] does. The second article builds on this and creates a distributed Intrusion Detection

System by carrying information flows marker on the network. Andriatsimandefitra and V. V. T. Tong, in their 2014 article [Andriatsimandefitra and V. V. T. Tong 2014], port to android these modifications done to the Linux kernel and then study the behaviour of malicious applications through the information dissemination observed during their execution.

As we can see, IFT can be implemented at different levels of abstraction. Each implementation has to balance flexibility and precision. Implementing a CPU-based IFT with flexible software policies is probably the most precise way to perform IFT. However, it is also very impractical. Our implementation of IFT is similar to those last articles. We use the data the Akheros Intrusion Detection System collects at the OS level while monitoring the system to infer the flows of information. This means that our solution can easily be installed on any system and run immediately. On the downside, since the flows of information are collected at the OS level, our approach will be less precise than hardware based approaches such as [Dalton, Kannan, and Kozyrakis 2007]. Since we lose precision with an OS level IFT scheme, we need a second level to validate the findings of the IFT. This is where we use a Hidden Markov Model. This stochastic model is a flexible representation of APTs; and by designing the appropriate score, we can use it to compare several potential APTs and select to one most probably an APT.

## 2.3 Hidden Markov Models

### 2.3.1 Presentation of hidden Markov models

A Markov chain is a stochastic process whose future state depends only on the current state. A Hidden Markov chain is a Markov chain which cannot be observed directly. The output of the chain, dependent on the current state, has to be observed instead. The states, which cannot be observed, are called hidden states, and the output of the states are called the observations. A Hidden Markov Model (HMM) $\theta$ is, thus, defined by three things: the state transition matrix $A = [a_{ij}]$ which describes the probability to go from any state to any state; the observation matrix $B = [b_{ij}]$ which describe the probability for each state to emit each output; and the initial state distribution $\pi = [\pi_i]$. HMMs can be used to solve three types of problems as described by Stamp in [Stamp 2004]. The first one is, given a sequence of outputs, to compute the likelihood of that sequence given the model, i.e was this sequence generated by the model. The second one is, given a sequence of outputs, to find the optimal sequence of states for the Hidden Markov chain, i.e. which sequence of states generated the sequence of observations. The third one is, given a sequence of outputs, to train a model to fit the data.

The naïve answer to the first problem, enumerating every state sequence, leads to a computationally unfeasible calculation even for short chains. A different approach is needed. The solution is the forward-backward procedure [Rabiner and Juang 1986; Devijver 1985], which basically walks along the trellis representing all the possible chains. During the forward half of the procedure, we inductively compute the probability of the partial observation sequence finishing with state $i$ at time $t$, $\alpha_t(i)$. The forward probabilities are initialised with

$$\forall i, \alpha_1(i) = \pi_i b_i(O_1),$$

where $O_t$ is the observation at time $t$. And then, we have

$$\forall j, \alpha_{t+1}(j) = \left[ \sum_i \alpha_t(i) a_{ij} \right] b_j(O_{t+1}).$$

The probability of the observation sequence of length $T$ is then $\sum_i \alpha_T(i)$. This method greatly reduces the number of computations needed to obtain the result. Similarly, the backward probability represents the probability of the partial observation sequence starting with state $i$ at time $t$. It is written $\beta_t(i)$ and is also computed inductively. It is initialised with $\beta_T(i) = 1$, and then

$$\beta_t(i) = \sum_j a_{ij} b_j(O_{t+1}) \beta_{t+1}(j).$$

This also reduces the number of computations needed to obtain the backward probabilities, which will be useful in other algorithms.

The Viterbi algorithm [Viterbi 1967; Forney 1973] gives the solution to the second problem which is finding the optimal sequence of hidden states given a sequence of outputs. It involves another walk on the trellis, this time storing, for each state, the most probable path to end there as well as its probability. It is initialised with

$$\forall i, v_1(i) = \pi_i b_i(O_1).$$

Then,

$$\forall j, v_t(j) = \max_i v_{t-1}(i) a_{ij} b_j(O_t).$$

We can then backtrack to find the corresponding states; the last state is

$$S_T = \arg\max_i v_T(i),$$

and then

$$\forall t, S_t = \arg\max v_t(i) a_{iS_{t+1}}.$$

This algorithm can be extended to find the $n$ most optimal sequences, with $n$ arbitrary as described by Seshadri and Sundberg in [Seshadri and Sundberg 1994]. The naïve parallel implementation of this idea requires $n$ time more storage and computation than the simple Viterbi algorithm. The article proposes a serial algorithm which requires $T$ times more storage and $T$ times the number of states more computations.

The third problem, that of training a model to fit the data, can be solved by the Baum-Welch algorithm [Baum and Petrie 1966; Sundaram 2000]. The algorithm uses the forward-backward procedure described above to compute the forward and backward probabilities and then uses these results to re-estimate the values of $A$, $B$, and $\pi$. First we compute two temporary values:

$$\gamma_i(t) = \frac{\alpha_i(t)\beta_i(t)}{\sum_J \alpha_J(t)\beta_J(t)},$$

and

$$\xi_{ij}(t) = \frac{\alpha_i(t)a_{ij}\beta_j(t+1)b_j(O_{t+1})}{\sum_I \sum_J \alpha_I(t)a_{IJ}\beta_J(t+1)b_J(O_{t+1})}.$$

The model can then be updated with:

$$\pi_i = \gamma_i(1),$$

$$a_{ij} = \frac{\sum_t \xi_{ij}(t)}{\sum_t \gamma_i(t)},$$

$$b_i(o) = \frac{\sum_{O_t=o} \gamma_i(t)}{\sum_t \gamma_i(t)}.$$

Starting from a model with random values for $A$, $B$, and $\pi$, these steps can be repeated until the model has sufficiently converged.

This algorithm, however, does not try to adjust the size of the model. For that task, other methods are necessary. There are a number of criterion which can be used to measure the performance of a given HMM, such as the Akaike Information Criterion [Akaike 1974] or the Bayes estimators presented by Schwarz in [Schwarz 1978]. H. Tong presents, in [H. Tong 1975], a method using the Akaike Information Criterion to find the order of a Markov chain. And Hernando, Crespi, and Cybenko presents, in [Hernando, Crespi, and Cybenko 2005], method to compute the entropy of the HMM for a given observation sequence, which reflects the uncertainty of the current model in tracking the hidden states. In order to be efficient, this method uses a trellis similar to the one used in the

Viterbi algorithm.

HMMs fit our situation well. We want to model the evolution of APTs but attackers do not inform us of what they are doing and why, so that, even when we detect an attack, we cannot be sure of the state of the APT. Instead, we can use the detected attacks as observations and use those to infer the evolution of APTs. The hidden part of the model becomes the different steps in an APT while the visible part are the types of attacks that have actually been found.

### 2.3.2 Different scores for HMMs

While the context of [Posada and Buckley 2004] is phylogenetics, the arguments presented by Posada and Buckley in favor of the use of the Akaike Information Criterion (AIC) and the Bayesian Information Criterion (BIC) do hold in the field of information security. The article starts from the observation that during model selection, none of the models represent reality, they are, rather, approximations of reality. However, likelihood ratio tests are instead looking for the model representing reality exactly. On the other hand, model selection using AIC or BIC can compare multiple models simultaneously and can take into account the model selection uncertainty. They also allow for model-averaged inference, which means that the inference is done using several of the best fitting models instead of using only the best one.

In [Biem, Ha, and Subrahmonia 2002], Biem, Ha, and Subrahmonia presents a new score specific for the selection of HMM topology. The score is based on the BIC, but points that the BIC does not account for the prior of the topology or the fact that parameters may not be homogeneous. Thus, they propose the HMM-oriented BIC (HBIC). HBIC starts from the BIC and adds terms to take into account the prior on the topology and the specific distribution of each parameter. They show that this score chooses better topology than standard BIC, achieving better results with simpler topologies. The same authors present another score in [Biem 2003] : the Discriminative Information Criterion (DIC). DIC is appropriate when selecting models representing classes of data. This time, the objective to optimise for raw performance without taking into consideration the size of the model. The differentiating characteristic of the DIC is that it takes into account the capacity of the model to generate data belonging to competing classes. In their tests, models selected using DIC had better recall at the price of increased model complexity.

Siddiqi, Gordon, and Moore presents, in [Siddiqi, Gordon, and Moore 2007] a new algorithm, STACS, for learning both the topology and the parameters of an HMM. STACS construct an HMM by alternating between parameter learning and model selection while

increasing the number of states. Candidates models are generated by splitting existing states and then optimising the parameters. They are then evaluated for selection using the BIC. The splitting algorithm is also novel because it takes into account both the contextual and the temporal structure of the data. The results show that STACS consistently returns larger HMMs than existing techniques. However, these HMMs also have higher classification accuracy.

MacKay Altman shows a graphical technique for assessing the goodness-of-fit of a stationary HMM in [MacKay Altman 2004]. Their technique involves plotting the estimated univariate distribution against the empirical univariate distribution. If the model is correctly specified, this plot will converge to the 45° line through the origin. In order to check the correlation structure, they also plot the bivariates distributions, which will also converge to the 45° line through the origin. The multivariates distributions (up to twice the number of hidden states in the model) can also be plotted to better check the fit of the model. However, the technique does not include a formal method of assessing the degree of variability in the obtained plots, i.e. whether the observed scatter around the 45° line is acceptable or not.

In [Titman and Sharples 2008], Titman and Sharples present a general goodness-of-fit test for both Markov and Hidden Markov Models. It is based on the work of [Aguirre-Hernández and Farewell 2002] which presents a Pearson-type goodness-of-fit for progressive Markov models. It generalise the test by taking into account absorbing states and censored observations. The test obtained is further extended to account for misclassified states. The final test is then used to assess the fit of data which could not be tested before. However, the test is not completely general but can be useful in cycles where the model is first estimated and then evaluated.

In [Flores, Antolino, and Garcia 2009], Flores, Antolino, and Garcia use HMMs to monitor the network for anomalies. They concentrate on the building of the HMMs. Instead of using the Baum-Welch algorithm, which cannot change the topology, they use genetic algorithms and show that they outperform Baum-Welch. They then improve their contribution in [Flores, Calderon, et al. 2015] by replacing genetic algorithms by evolutionary programming, and by also adding the time as one of the inputs of the HMMs. Switching to evolutionary programming enables better modeling of the HMMs for evolution which, in turn, improves the exploration of the solution space. By adding the time, the HMMs are now capable of identifying that a pattern may be normal during certain times and abnormal during others. Once again, the models obtained were more accurate than those obtained with Baum-Welch, and they were also more accurate than those created by experts.

As we can see, there are a number of scores available when trying to select the best

performing model. However, we are trying to compare the fit of different chains of potential APTs on the same model. We have not found any score appropriate for this task which means we will have to create our own score.

### 2.3.3 HMMs used in security

Hidden Markov model have already been used for intrusion detection. Ye, Y. Zhang, and Borror, in their 2004 article [Ye, Y. Zhang, and Borror 2004] present a framework for using Markov chain models to detect anomalies. The Markov chain represents the normal usage of the system, and any attack or anomaly should make the behaviour of the system deviate from the norm. The states considered in the Markov chain are the audit events from the system. To reduce the amount of computation, only the window of the last N events is considered when computing the fit to the model. This article shows that this Markov model approach works better for smaller windows and if the dataset is cleaner. It also shows that Markov models can be used to model the baseline behaviour of computer systems.

In [Ioannou et al. 2013], in 2013, Ioannou et al. maps the phases of an APT to a cyber attack kill chain. A Markov multi-phase transferable belief model is then used to fuse incoming data from a variety of sources. It is used to show what the attacker has been doing and also predict their next steps. The model is constructed phase by phase. First, a number of attacks are assigned to a given phase, then each attack is linked to every attack in the previous phase which could enable the attacker to perform the current attack. This forms a tree. Then, when an attack is detected, the nodes it corresponds to on the tree and their subtrees are kept and the rest are pruned. The same is done when the next attack happens, and, as more and more attacks happen, the tree becomes more and more sparse and the next step in the APT can be predicted with a better precision. This approach is close the what we want to do. It shows that an Markov chain can be used to model attack campaigns. However, the preliminary steps of listing and linking every attack possible is not reasonable, and we think it could be avoided by using Hidden Markov chains.

Jain and Abouzakhar, in their 2012 article [Jain and Abouzakhar 2012] show an example of using Hidden Markov chains to model attack chains. It focuses on the network and uses Hidden Markov chains to model TCP services. Models are trained for each TCP session for each service. They are then used to detect anomalies. This article focuses on the learning phase, and more specifically on the feature selection. Features are selected using a J48 decision tree. This leads to a substantial reduction in error rates and an increase in detection rates. This article shows that HMM can be used to model

scenarios such as protocols.

Ourston et al. attempts to detect multi-stage attacks using a HMM, in [Ourston et al. 2003] in 2003, and compare its performance to that of decision trees and neural networks. The article explains that HMMs are suited to multi-steps attacks because each step is dependent on the outcome of the previous state. In their experiments, they show that HMMs outperform both decision trees and neural networks even with a low number of training example. This exacerbated by what they call the "rare data problem", i.e. the fact that the most dangerous attacks happen the least often and are thus hard to train for. Q. Zhang, Man, and Wu base the work they do in [Q. Zhang, Man, and Wu 2009] on this article in order to discover the intent of the attackers. Their first assumption is that each sensor in a system could be mistaken. To compensate for this, they create a HMM for each sensor and then apply a decision algorithm on the output of the HMMs and obtain a guess of the attacker's intent. Both of these articles show that HMMs are effective for modeling the evolution of attacks. However, they do not take into account the possibility of having several attack campaign active at once, in which case the HMM would evaluate the evolution seen as if there was only one attack campaign.

The aim of [Arnes et al. 2006] in 2006 is to assess the impact of certain events on the security status of an entire network. To do so, Arnes et al. use a HMM to model the state of each system in the network and the transitions between states. The HMM contains four states: "good", "probed", "attacked" and "compromised". Each system in the network is given a cost vector, which represents the cost of having the system in each of the state at given time. The cost vector and the HMM are used to compute the risk factor of the host. The total risk for the network is the sum of the individual risks of each system. This shows that HMMs can be used to model the state of a system without making assumptions about the transitions between attacks.

In 2006, in [Khanna and H. Liu 2006], Khanna and H. Liu present a system to detect intrusions in ad hoc networks. It collects data about the network, clusters and classifies it to create observations and then use a profile estimator to estimate the state of intrusion of the system. A HMM using the same states and observations is then used as a feedback loop for the profile estimator and to compute the drifts of the different profiles. The states considered are "normal", "hostile intrusion attempt", "friendly intrusion attempt", "intrusion in progress" and "intrusion successful". The authors refine this approach in [Khanna and H. Liu 2008]. The number of states is reduced to three: "normal", "intrusion in progress" and "intrusion successful", and a PID controller is added to the feedback loop of the profile estimator to reduce the rate of false positives.

In 2007, Haslum, Abraham, and S. Knapskog present their article [Haslum, Abraham, and S. Knapskog 2007] on a distributed network intrusion prevention system which in-

cludes a HMM in order to make predictions about attacks. The HMM contains four states, "normal", "intrusion attempt", "intrusion in progress" and "successful attack". The results of the model are used to assess risk using fuzzy inference systems. In [Haslum, Moe, and S. J. Knapskog 2008], they switch to a Continuous Time HMM (CTHMM) in order to model the interactions between the attacker and the monitored system. They keep the same states. This approach results in a lower false positive rate from the IDS. This is even further improved in [Haslum, Abraham, and S. Knapskog 2008], where there is now a different HMM for each sensor. The state of each individual HMM as well as the state of the global HMM are all used to determine the risk level using fuzzy logic.

Recent attacks contain anti-forensics measures targeting the identification and acquisition phases of forensic analysis. Maggi, Zanero, and Iozzo, in [Maggi, Zanero, and Iozzo 2008] in 2008, propose to build Markov models of processes based on the sequences of system calls they use. Profiles need to be learnt for each application in advance. Then, the live system is monitored and each system call is analysed. Two probabilities are computed, the probability that the execution sequence fits in the model and the probability of the system call appearing at that point in the sequence. These probabilities are compared to the anomaly threshold in order to decide if anti-forensic measures are being used.

In 2012, in [C.-M. Chen, Guan, et al. 2012], C.-M. Chen, Guan, et al. feed heterogeneous logs to a three-state HMM in order to detect attacks against machines in the cloud. The states are "reconnaissance" where the machine is being probed, "intrusion" where the machine is being compromised and finally "attacking" where the machine is being used to attack other machines. The authors use a similar approach in [C.-M. Chen, Hsiao, et al. 2013] to protect Industrial Control Systems (ICS). The aim is to detect attacks where the target is first attacked by a botnet trying to find credentials. Once credentials are found, a human attacker connects to the target and makes it part of the botnet, which means it will be used in subsequent attacks. Such a scenario is possible because ICS often use weak password susceptible to brute force attacks. In this case, the HMM used has three states: "normal" when the target is not under attack, "intrusion" when the target has been compromised and "joint attack" when the target is used to perpetrate further attacks. In [C.-M. Chen, P.-Y. Yang, et al. 2014], the authors focus even more on finding the link between the bruteforce attack by a botnet to find the password and the attacker using that stolen password from a different machine. In order to differentiate an attacker using a stolen password from a normal user login, their proposed system monitors the network for passwords being exfiltrated. Alerts are raised whenever a bruteforce attack is detected and whenever an exfiltrated password is being used. These alerts are sent to a HMM which differentiates between an attacker and a normal user. These re-

sults are interesting to us because we want to use HMMs to model the links between attacks in an attack chain.

While Katipally, L. Yang, and A. Liu does not mention APTs in their 2011 article [Katipally, L. Yang, and A. Liu 2011], their aim is to analyse the behaviour of attackers during multi-stage attacks which is a more general vision of APTs. They define five categories of attackers such as the criminals trying to make money, the terrorists trying to destroy and the insider with knowledge of the organisation. They then use one Hidden Markov Model for each kind of attacker, and are able to classify attackers into one category. The actual defense can then be adapted depending on the category. The models use five phases for attacks: scanning, enumeration, and then three types of exploitation, access, malware or denial of service. This is really interesting, because it shows that HMM are precise enough that they can be used to classify different kinds of attackers.

Even earlier, in 2003, in [Gao, Sun, and Wei 2003], Gao, Sun, and Wei explore the capabilities of HMM to make predictions about intrusions. They use Hidden Markov chains to model application-level protocols. They show that these models can be used to predict attacks before the full payload is seen. However, during model creation, each application must be analysed manually in order to extract interesting features. Even though the training can be automated, and the results show that attacks can be predicted, the fact features have to be extracted manually makes this solution less automated than it should be.

As we can see, HMMs can be used to model attack campaign. However, none of these experiments take into account the possibility that there are several active attack campaigns in a system at any given time. This is not acceptable when working on the detection of APTs as those are usually long-lasting campaigns which increases the odds that other campaigns will be active at the same time. This is why we use IFT to reconstruct potential attack campaigns and then use HMMs to evaluate these potential attack campaigns. In addition, since APTs are the work of skilled attackers, not every attack will be detected. This means that our usage of HMMs must be able to take into account possibly missing observations when evaluating whether a given chain is an APT.

## 2.4 Data generation for machine learning and security

Machine learning approaches often require large datasets for training and validation. In security, there are public datasets available for the evaluation and testing of Intrusion Detection System. The most well known is the IDEVAL dataset from 1999 [Lippmann et al. 2000; Haines et al. 2001]. This is a comprehensive dataset, including data for the

evaluation of both network-based and host-based IDS. However, it is not without flaws, highlighted in numerous articles such as [McHugh 2000; Mahoney and Chan 2003; Engen 2010]. Despite its age and inherent shortcomings, it is still being used [Z. Chen et al. 2016; Little, Mountrouidou, and Moseley 2016]. There have been proposal to fixes the flaws and update the dataset, such as the one by Qian, Xu, and Shi [Qian, Xu, and Shi 2006], but a 2014 study by Koch, Golling, and Rodosek [Koch, Golling, and Rodosek 2014] shows that available dataset are still not satisfactory. The one cited as the most realistic is the PREDICT, recently renamed to IMPACT [DARPA 2016], however, it is not a public dataset. Hence, the community requires better public data in order to properly evaluate and compare IDS solutions. This is compounded by the results of Maxion and Tan [Maxion and Tan 2000] which shows that the performance evaluation of an IDS can differ by an order of magnitude simply by changing the dataset.

Cordero, Vasilomanolakis et al. have worked on the creation of datasets for the evaluation of IDS [Cordero et al. 2015; Vasilomanolakis et al. 2016; Vasilomanolakis 2016]. They start by identifying four requirements that make a dataset qualified for the evaluation of IDS. The dataset must contain labeled attacks. The dataset must contain modern normal and attack data. The dataset must be publicly available or reproducible. The dataset must be flexible and allow testing different scenarios. In this light, they propose ID2T, a tool generating labeled datasets from network packet captures while making sure to retain realistic properties. The tool is interesting, and the datasets produced can be used to evaluate any network-based IDS. However, this is limited exclusively to network-based IDS.

In 2015, Małowidzki, Bereziński, and Mazur start by listing and analysing the datasets used in articles, when they are disclosed. Then, in [Małowidzki, Bereziński, and Mazur 2015], they identify the two main shortcomings of these datasets. Namely, they are often stale and often lack labeled attacks. From these shortcomings, they define four criteria as essential to a good dataset. It must be recent. It must include attacks labeled with precisions. The attacks must be varied. And the dataset must be correct. They also add the bonus criterion of a balanced dataset, containing both attacks as well as normal data. They analyse existing datasets and come to the conclusion that the best datasets are collected in actual production environment.

Datasets collected in production environments are rare and usually not public. Instead, we are interested in tools to generate the datasets. These tools should be available so that anyone can generate the datasets. Such tools already exist and we will describe a few.

Béla Genge et al.; Siaterlis, Béla Genge, and Hohenadel focus, in [Béla Genge et al. 2012; Siaterlis, Béla Genge, and Hohenadel 2013], on Industrial Control Systems (ICS).

They propose an environment for experiments on the security of ICS. They identify generic requirements such as reproducibility, automation and controlled environment. There are also ICS specific requirements such as a wide range of physical processes, a wide range of ICS or at least the typical ones, real software support so that actual SCADA and malwares can be run, accurate representation and safe disruptive experiments. The last one is particularly important for ICS as these are systems which act on the physical world, and attacks can cause unpredictable results. It is necessary that these unpredictable results do not cause harm or cripple the experiment framework. Their solution is to have physical models for the real world interactions and simulate them during the experiments.

MACE [Sommers, Yegneswaran, and Barford 2004] is a 2004 framework for recreating malicious packet traffic by Sommers, Yegneswaran, and Barford. This is done by defining a model of the malicious traffic. The model contains three parts. The exploit model, which describe the vulnerabilities part of the attack. The obfuscation model, which details how the payloads are morphed in order to avoid detection. The propagation model, which details how the victim are chosen and in which order. There is an additional model describing the background traffic which represents the normal usage of the network. MACE then uses these models to generate network traffic which can be analysed by whatever tool is being developed. However, the tool is not publicly available because the authors fear that it could be used for malicious purposes.

In 2006, Mirkovic et al., in [Mirkovic et al. 2006], design a benchmark for Distributed Denial of Service (DDoS) defense evaluation. The benchmark can be configured on three points: the type of DDoS attack, the legitimate traffic and the network topology. The type of DDoS attack is important because depending on the attack, the network packets and the traffic shape will be completely different. The legitimate traffic is important because it may or may not resemble the malicious traffic, which means that it can be more or less easy to pick it out and allow it through. Lastly, the network topology also has a big impact on the effectiveness and the architecture of the DDoS defense. These settings are defined from a set of collected DDoS traces in order to offer a collection of typical DDoS attack scenarios. The benchmark also includes performance metrics and measurement methodology so that the results are more directly comparable.

These tools are interesting, but we find that they do not offer enough variety in the datasets they can generate. The first one is focused on ICS, the second on network traffic and the third on DDoS attacks. In light of this, we propose our own tool, Moirai [Akheros 2016a]. This tool does not actually collect data to create a dataset. Instead, this tool is meant to replay scenarios. Whatever IDS is to be tested can be installed on the appropriate VMs in the scenario and will collect the data. The tool is described in more details in Section 5.2 on page 81, and the scenarios used with it to evaluate our solution

in Section 3.2 on page 50 and Section 4.2 on page 68 are available [Akheros 2016b].

## 2.5 Summary

We have seen that APTs are a dangerous threat and that they are difficult to detect. While there are works trying to do just that, there is no definitive solution. The tool we present hereafter to reconstruct APT campaigns builds on the findings of these earlier solutions and in particular uses the same decomposition in phases of attack campaigns: "reconnaissance", "compromise", "establish presence", "privilege escalation" and "mission completion". We will then use IFT to link separate attacks together in potential attack campaign. The intuition is that in an attack campaign, the attacker uses one phase of an attack in order to set up the next. This means that there should be flows of information from the first attack to the second one. Using IFT, we can track those and thus reconstruct attack campaigns. However, IFT also has a lot of false positives and we need to filter those out. To do this, we use an HMM, with a custom made score in order to be able to compare the fit of different chains to the model, even if some of these chains have missing observations. HMMs fit our use case because the phases of an attack campaign are the values we are interested in; however, they are not directly observable. Instead, we can deduce the phase of an attack campaign by the type of attack that is observed. Since the other Akheros modules detect and qualify individual attacks, we can use those qualified attacks as the observations. Additionally, attack campaign should fit the Markov hypothesis rather well since the actions of an attacker depend on whether the current attack succeeds. If the current attack succeeds, the attacker will move on to the next phase; if it fails, they will either stay in the same phase, or they will go back to a previous phase. In either case, the progression of the campaign is only dependent on the result of the current attack. Our work on IFT will be described and evaluated in Chapter 3 on page 39 and the work on HMM is detailed in Chapter 4 on page 55.

In addition, we have seen that, while there all datasets to evaluate IDS with, they are heavily criticised. We decided to create our own tool, Moirai, which concentrates on defining and replaying scenarios. We use it to create APT campaigns and assess how our IFT and HMM based attack campaign reconstruction and evaluation tool performs. Moirai is detailed in Chapter 5 on page 77, accompanied the description of the rest of the supporting architecture we developed.

# Chapter 3

# Information Flow Tracking – Finding links between attacks

---

*In this chapter, we describe the IFT system used to link attacks together in order to build chains of attacks. We list the containers of information, files, processes and sockets, and explain how we extract flows of information from the raw data. We then add a taint whenever a new attack is detected and use those taints to link attacks together: whenever a taint is on the inputs of a flow part of an attack, this means there is a link between this attack and the attack represented by the taint. This is how, by the end of this chapter, we will have extracted potential attack campaign. However, this is only the first of two steps as our implementation of IFT leads to over-approximations. A second step, explained in the next chapter, is needed to filter the results.*

---

## 3.1 Method

### 3.1.1 Concept presentation

When programs run in an information system, they exchange data with the system and with other programs. The idea behind Information Flow Tracking is to track those data exchange in order to follow the data. The aims can differ, but an obvious example would be to check that some specific data does not reach some specific place, e.g. that sensitive data is not sent on the network. The flows of information can be tracked at different levels of granularity, with finer granularity requiring more resources but giving more precise results. In order to track these pieces of information inside the system, IFT implementations attach some metadata, usually called taint or tag, to these pieces of

information. This metadata indicates through which components of the system the data has travelled. We can then apply policies on the metadata in order to safeguard the data. Taking the same example as before, we would have a specific taint for sensitive data and specify that data with this taint should not be sent through a network device. As the system run, some programs would handle sensitive data, and any data derived from this sensitive data would also gain the "sensitive" taint. If any program were to send data with the "sensitive" taint on the network, then the IFT implementation would raise an alert. This would happen regardless of how the "sensitive" taint arrived on the data. In particular, the IFT implementation does not care when and how taint are attached to the data; this means that IFT implementation do not need to know how a given taint could reach a given point of the system, it only needs to track flows of information in order to protect the system.

Concretely, an IFT implementation must define a number of concept. The first one is the granularity of the tracking; this is defined through the concept of "containers of information". Containers are the smallest unit of data that can be tracked. Depending on the implementation, it can be a byte or it can be a whole file or program. The second concept are the taints, and more specifically, how taints are added inside the system. This could mean taints based on the location of the original data for example. Lastly, the implementation must define rules for propagating the taints inside the system. This is similar to defining the flows of information as well as defining the rules governing how taints are merged on containers, i.e. what happens when a container that already contains tainted data receives data with a different taint.

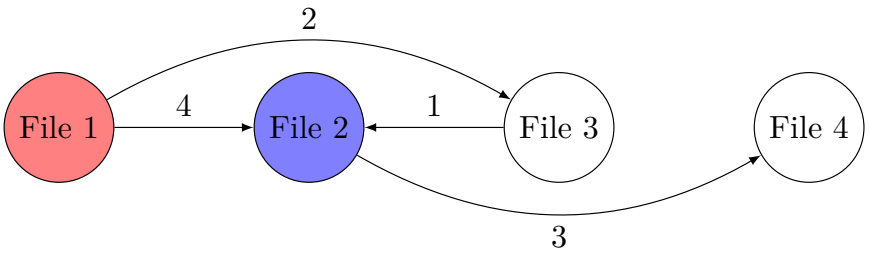
Let us present an example based on our usage of IFT and illustrated in Figure 3.1 on page 42. We start by enumerating all the containers. In this example, the containers are files on a computer and there are four of them. We then add the initial taints to the appropriate containers. In this case, we want to see how the data in two files is being used so we give a unique taint to File 1 and another one to File 2. We define the rules of propagation as a simple union of the taints already present on the container with the new taints added from the flow of information. The last stage of the initialisation is adding the flows of information, in chronological order. In this example, there are four flows of information which are created when a program reads from one file and writes to another. The result is 3.1a on page 42. We can then start propagating the taints. The first flow goes from File 3 to File 2, but since File 3 does not have a taint, nothing is propagated to File 2 and File 2 just keeps its taint, as shown in 3.1b on page 42. The second flow propagates the taint of File 1 to File 3. Since File 3 did not have a taint, it simply takes the taint of File 1, as shown in 3.1c on page 42. Similarly, the third flow puts the taint of File 2 on File 4 as shown in 3.1d on page 42. The fourth flow propagates the taint

of File 1 on File 2. Since File 2 already had a taint, its new taint is a union of both, as shown in 3.1e on the following page. We now know from the final taints that File 2 and 3 contain information originating from File 1 and that File 4 contains information from File 2. There is no way to know that File 2 contains information from File 3 because we did not give a taint to File 3; this is fine because we do not care about the information in File 3 to begin with. The exact propagation rules depend on the implementation and the requirements. For example, the rules could define that in the fourth flow, the new taint of File 2 is only the taint of File 1 instead of a combination of both taints. A security system using IFT could define rules to be satisfied in the system, such as specifying that File 4 cannot contain information from File 1 or that information from File 1 and File 2 should not be mixed. Those rules can be checked after each information flow and an alert raised whenever one rule is violated.

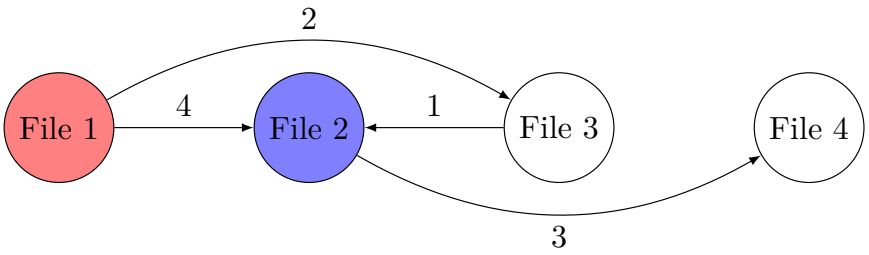
### 3.1.2 Definitions

The IFT module receives as input the data from an Intrusion Detection System (IDS) which also captures the low-level data required for tracking flows of information. The data it receives is composed of events in chronological order. Some of these events are marked as being part of an attack. Hence, an attack can be seen as a list of events such as is shown in Table 3.1 on page 43. Some of the types of events can also be interpreted as flows of information. The list of such events is in Table 3.2 on page 43. Each line in the table indicates that information is passed from the input objects of the event to its output objects. Most events only have one input and one output object, but some have more. For example, the "Read file" events indicates that the current process reads a file. In terms of information flow, this means that there is a flow of information from the file, the input object, to the process, the output object. As such, events are considered as tuples with four values: the timestamp, the event type, the list of references of input objects and the list of references of output objects. If either of the list of objects is empty, then the event cannot be interpreted as an information flow. These events do not appear in Table 3.2 on page 43. Objects are the elements of the monitored system which either contain data, the files and processes, or are communication channels, the sockets; information flows from and to objects.

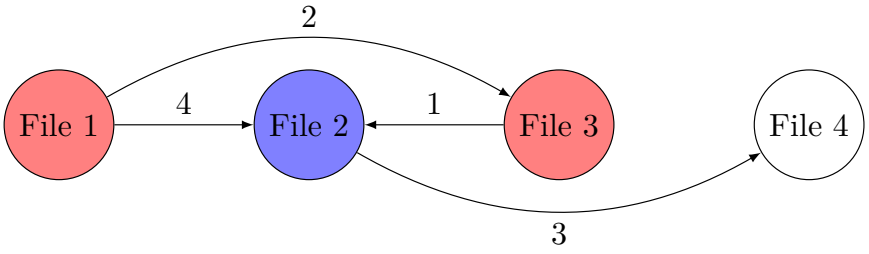
By processing these events in chronological order, our IFT system is able to reconstruct the flows of information through the monitored system. Note, however, that there is no way for our Information Flow Tracking system to know how the data read by processes are handled. The system has to make an over-approximation and assume that any information coming out of a process, or any other object, is a function of all the information that went into the process before. Thus, the system assumes there is a flow of



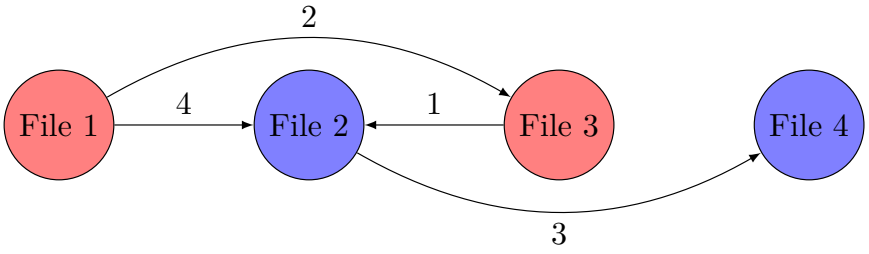

(a) Initial setting, with containers, taints and flows.



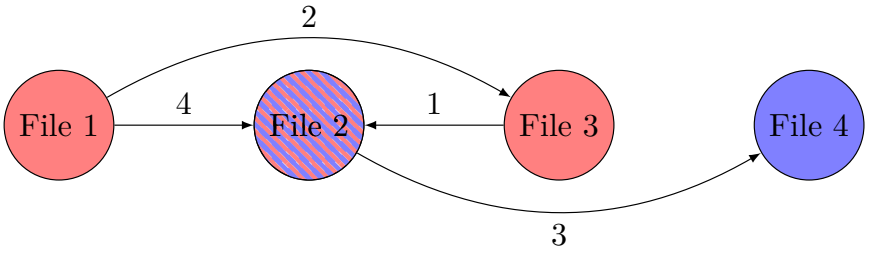

(b) Propagating the taint of flow 1.




(c) Propagating the taint of flow 2.




(d) Propagating the taint of flow 3.




(e) Propagating the taint of flow 4.

Figure 3.1: An example of taint propagation.

Table 3.1: A sample of the input for Information Flow Tracking

| Timestamp | Type of event | Input objects | Output objects | Attack |
|---|---|---|---|---|
| 1458208373036307835 | open file | – | – | – |
| 1458208376131582000 | receive packet | 10.0.51.100: 38201 | 10.0.51.101: 80 | – |
| 1458208376133138842 | close file | – | httpd (1184) – | – |
| 1458208376131582000 | receive packet | 10.0.51.100: 38201 | 10.0.51.101: 80 | Yes |
| 1458208376139787235 | read file | /var/www/cgi-bin/shell.sh | httpd (1184) httpd (1184) | – |
| 1458208378187978439 | write file | httpd (1184) | /tmp/pwnme | – |
| 1458208379179691846 | execute file | /tmp/pwnme | pwnme (1463) | Yes |

Table 3.2: List of events and their information flows

| Type of event | Input objects | Output objects |
|---|---|---|
| Create process | current process | new process |
| Execute file | filename | current process |
| Change permissions | current process | filename |
| Change owner | current process | filename |
| Create file | current process | filename |
| Read file | filename | current process |
| Write file | current process | filename |
| Receive packet | remote socket | local socket current process |
| Send packet | local socket current process | remote socket |

information between all the information that entered the object before an information comes out and that information coming out.

### 3.1.3 Implementation

Working from the list of events from the IDS and the detected attacks, we aim to highlight chains of attacks. As already mentioned, our work is based on the intuition that if two attacks are consecutive steps in an attack campaign, then there must be a flow of information from the outputs of the events part of the first attack to the inputs of the events of the second attack. For example if the first attack installs a RAT, the second attack can use that RAT to exfiltrate sensitive data. When the first attack installs the RAT, there will be an information flow from the attack to the process running the RAT. Then, when the second attack uses the RAT, there will be an information flow from the RAT to the data exfiltration. Thus, in order to link attacks and recreate attack campaign, we need to know which flows of information were part of which attack. The solution is to create a tag for each new attack. These tags are immutable and represent one attack, and each attack is represented by one tag. By propagating these tags through the flows of information and attaching them to objects, we will be able to follow where the information from attacks goes.

The tags representing each attack are propagated by the flows of information. Each flow of information is processed in chronological order. The propagation depends on whether the inputs are tagged and whether the event is part of an attack. The instructions are summarised in Algorithm 1 on the facing page:

1. The inputs are not tagged and the event is not part of an attack: there is nothing to do;
2. the inputs are tagged and the event is not part of an attack: the tags on the inputs are propagated to the outputs (Propagate);
3. the inputs are not tagged and the event is part of an attack: a new chain of attacks containing only this attack is added to the list of chains of attacks (CreateChain) and the tag representing the attack is propagated to the outputs (Propagate);
4. the inputs are tagged and the event is part of an attack: the attack is appended to the appropriate chains of attacks (AppendAndMergeChains) and the tag representing the attack is propagated to the outputs (Propagate).

As an example, we can apply this algorithm to the events in Table 3.1 on the previous page. The propagation is represented in Figure 3.2 on page 46. First, we establish the list of objects, of which there are 6 in this example. We can then start to insert and

**Algorithm 1** Rules of tag propagation

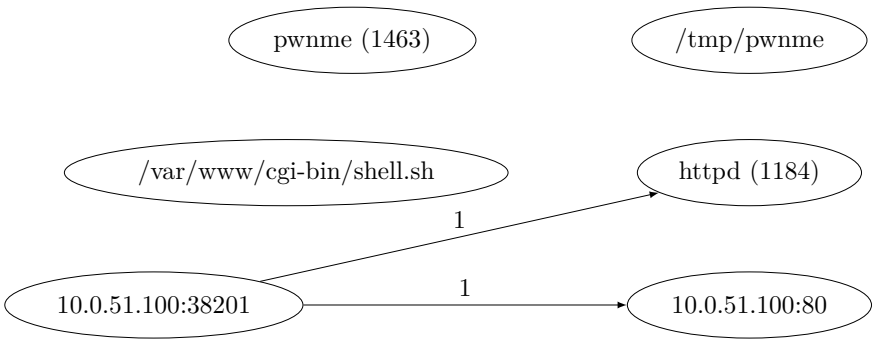
```
if NOT event is part of an attack then
    if NOT inputs are tagged then
        continue
    else
        Propagate(inputs, outputs)
    end if
else
    if NOT inputs are tagged then
        CreateChain(attack)
        Propagate(attack tag, outputs)
    else
        AppendAndMergeChains(inputs, attack)
        Propagate(attack tag, outputs)
    end if
end if
```

propagate tags. For the first flow of information, there are no tags in the system and no attacks so no objects are tagged, as shown in 3.2a on the next page. The second flow is part of an attack. Since there are no pre-existing tags in the system, we create a new tag and propagate it to the outputs of the flow as in 3.2b on the following page. In particular, we see that the process `httpd (1184)` is tagged with the tag of the first attack. We continue propagating tags, as shown in 3.2c on the next page. We see that `httpd (1184)` writes to the `/tmp/pwnme` file which becomes tainted with the tag from the first attack. This file is then executed, which is detected as a second attack. This means that the resulting process is tagged with the tag of this second attack. Additionally, a link is established between the first and the second attack.

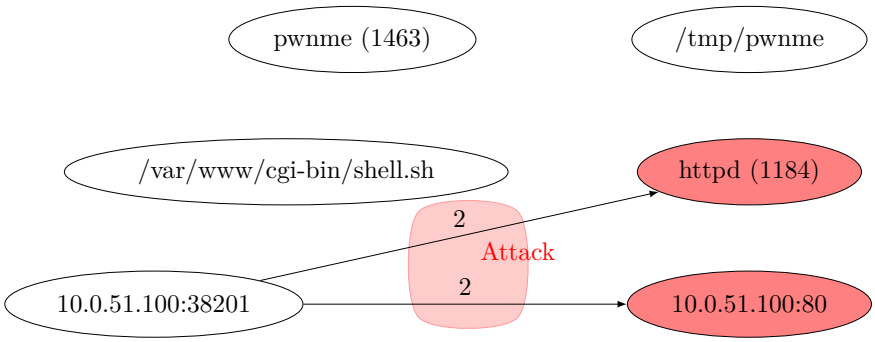
The Propagate function merges the tags already present on the outputs with the tags on the inputs as in Equation (3.1), where **I** is the set of inputs and **O** is the set of outputs.

$$\forall O \in \mathbf{O},\ \mathrm{Tags}(O_{t_n}) = \mathrm{Tags}(O_{t_{n-1}}) \cup \bigcup_{I \in \mathbf{I}} \mathrm{Tags}(I_{t_{n-1}}) \tag{3.1}$$

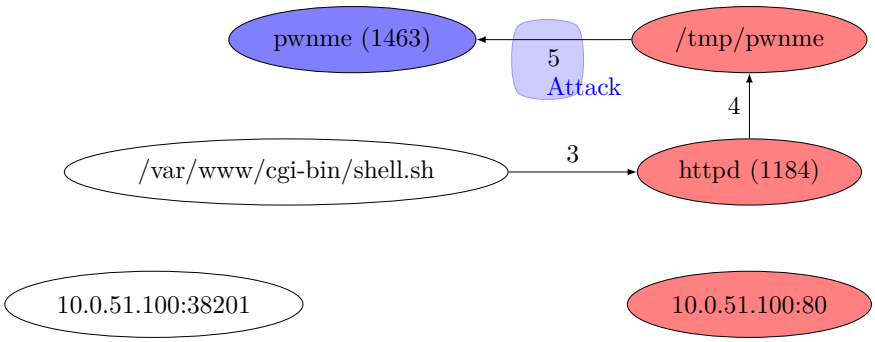
The AppendAndMergeChains puts the latest attack in the appropriate attack chains. In some cases, this includes merging separate chains of attacks that become linked by this new attack. It proceeds as follows. Let $L_{CA}$ be the list of chains of attacks ( Equation (3.2) on page 47). First, all the tags of the inputs of the new attack are put in a set, $T$ ( Equation (3.3) on page 47). Then, each chain in $L_{CA}$ finishing with a tag in $T$ is moved to $L_{new}$, a new list ( Equation (3.4) on page 47). Then, each chain in $L_{CA}$ containing at least one tag of $T$ is copied; each copy is truncated after the last tag




(a) Propagating the first flow.




(b) Propagating the second flow.




(c) Propagating the rest of the flows.

Figure 3.2: Propagating taints and finding links between attacks in our IFT implementation.

$$L_{CA} = \{A_1 - A_4, A_2 - A_5, A_3\} \tag{3.2}$$

$$\text{new attack: } A_6,\ T = \{2, 4\} \tag{3.3}$$

$$\begin{aligned} L_{CA} &= \{A_2 - A_5, A_3\} \\ L_{new} &= \{A_1 - A_4\} \end{aligned} \tag{3.4}$$

$$\begin{aligned} L_{CA} &= \{A_2 - A_5, A_3\} \\ L_{new} &= \{A_1 - A_4, A_2\} \end{aligned} \tag{3.5}$$

$$\begin{aligned} L_{CA} &= \{A_2 - A_5, A_3\} \\ L_{new} &= \{A_1 - A_2 - A_4\} \end{aligned} \tag{3.6}$$

$$\begin{aligned} L_{CA} &= \{A_2 - A_5, A_3\} \\ L_{new} &= \{A_1 - A_2 - A_4 - A_6\} \end{aligned} \tag{3.7}$$

$$L_{CA} = \{A_1 - A_2 - A_4 - A_6, A_2 - A_5, A_3\} \tag{3.8}$$

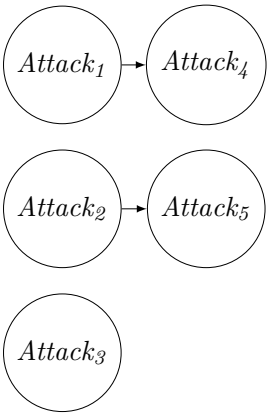
---

of $T$ in it; the truncated copies are put in $L_{new}$ ( Equation (3.5)). Then, all the chains in $L_{new}$ are merged by creating a new chain with all the attacks in chronological order ( Equation (3.6)); the new attack is appended to this chain ( Equation (3.7)); this new chain is added to $L_{CA}$ ( Equation (3.8)).

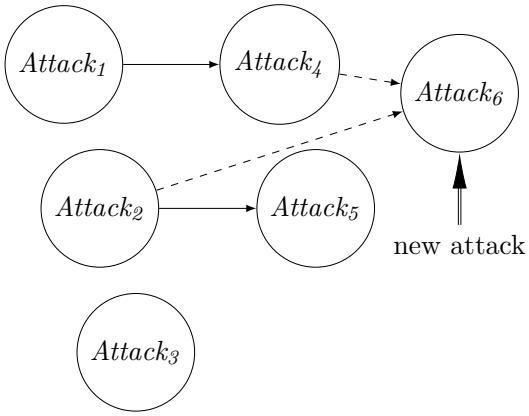
This is also illustrated in Figure 3.3 on the following page. The initial $L_{CA}$ is shown in 3.3a on the next page. $Attack_6$ is detected and linked to $Attack_2$ and $Attack_4$ in 3.3b on the following page. The chain $Attack_1$ – $Attack_4$ is removed from $L_{CA}$ and added to $L_{new}$. The chain $Attack_2$ – $Attack_5$ is copied and truncated, leaving only $Attack_2$ and added to $L_{new}$. The chains in $L_{new}$ are then merged and added to $L_{CA}$ giving the results in 3.3c on the next page.

### 3.1.4 Pros and cons: why we use Information Flow Tracking

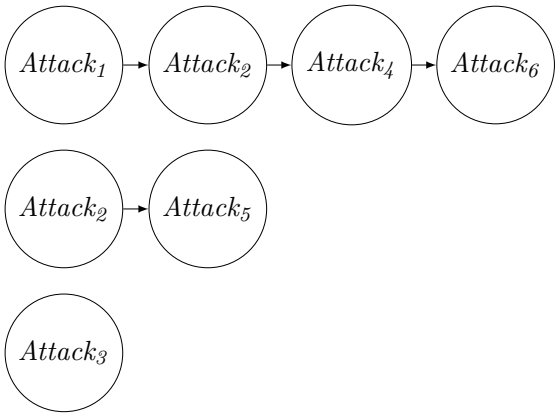
The principles behind IFT can seem simple: if a given bit of information is copied from one place to another, then there is a flow of information between those two places. However, this is only the simplest case of flow: a direct flow of information. There are also indirect flows of information, where the information contained in a given bit is not copied to a second location, but instead that first bit influences the value that is being copied to the second location. An example is the first bit being used as a condition in execution branches. Depending on its value, the program will take one of two branches and the value of the second location, set during these branches, will be different in each




(a) Chains of attacks before the new attack is detected.




(b) A new attack is detected and new links are created.




(c) Chains of attacks are updated with the new attack.

Figure 3.3: AppendAndMergeChains in action.

case. There is an implicit flow of information from the first bit to the second one. Thus, IFT is capable of following flows of information even if the attack only modifies the logic, and not the data, of the attacked program.

If we consider a perfect IFT system, it is able to follow the changes of any bit in the system. This means that if it finds that the information contained in a number of bits originated in one attack and travelled through the system until it was used in another attack, then those attacks are linked. This means that the second attack used information originating from the first attack, which means they are almost certainly linked. This is the reason we use IFT. If it finds a link between two attacks, then there is an actual physical link between these two attacks. However, this link does not immediately translate to those two attacks being part of the same attack campaign. For example, the second attack could be attacking a program that was modified by the first attack. This is compounded by the fact that if a service is vulnerable, several attackers could take advantage and use this same service as an entry point, which, in some cases could create links between the attacks. Thus, while IFT does show a physical link, it cannot know the cause of the link and cannot infer the intent of the attackers. This can create spurious links between attacks.

Our implementation of IFT is not perfect. Creating a perfect IFT system would require hardware assistance and incur a high overhead as a lot of computations are necessary to follow the information flows of every single instruction. Hence, our implementation has an additional drawback. As discussed already, it has to make over-approximations because it cannot follow information at the granularity of the size of a bit. Instead our implementation works at the granularity of the files and processes. Every bit in a file has the same tags, every bit in a process has the same tags. So, once a process reads a file with a tag, it will stay tagged until it dies, even if the part of memory where the content of the file was stored and processed is removed. In this case, if a service is vulnerable and the target of several attacks, these attacks will always be linked by IFT. Even if the service processes the data from the attack and then discards it before the next attack, it remains tagged which links the two attacks together.

Another interesting property of IFT is that it is resilient in case we miss attacks. If the flows from $Attack_1$ lead to $Attack_2$ which then lead to $Attack_3$, what happens if the IDS does not detect $Attack_2$? There are actually two cases to consider, and both are illustrated in Figure 3.4 on page 51. In the first case, the flows linking $Attack_1$ to $Attack_2$ are parents of the flows linking $Attack_2$ to $Attack_3$, as in 3.4a on page 51. In this case, if $Attack_2$ is not detected, since the flows from $Attack_1$ are parents of the flows from $Attack_3$, IFT will establish a link between the two attacks, as in 3.4b on page 51. This is what we mean when we say that IFT is resilient in case we miss attacks. There

is also a second case to consider. We have defined attacks as a list of events. This means we can have a configuration of events as shown in 3.4c on the next page. The flows of $Attack_2$ which are parents of flows of $Attack_3$ are not descendants of flows of $Attack_1$. In that case, if we miss $Attack_2$ then we end up with the configuration in 3.4d on the facing page where there are no links established between $Attack_1$ and $Attack_3$. In that case, IFT is not resilient. However, in our experiments, we studied the behaviour of IFT when the IDS requires only one event to detect an attack. This is the worst case for IFT when trying to establish links as attacks made of more events will lead to broader diffusion of tags, but this matches the first case which means that for our application, we can consider that IFT is resilient to missing attacks.

All in all, IFT is not perfect but it does highlight physical links between attacks. And since our implemented system integrates with an IDS that already collects all the necessary information, there is no additional cost in adding IFT. Of course, that means IFT is only the first step in finding attack campaigns. Subsequent steps are necessary to filter out false positives in the links created.

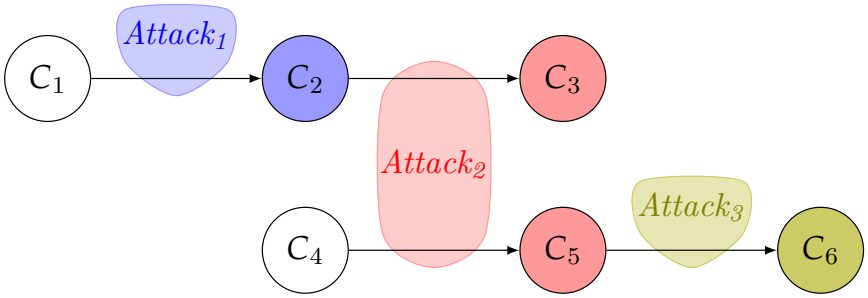
## 3.2 Evaluation on APTs

In this evaluation, we use the scenario presented in Chapter 5 on page 77. This scenario is a naïve but representative example of an APT: we include all the usual steps in the right order but exclude any cycle. Figure 5.3 on page 87 gives a visual overview. It follows a website encountering four different types of users: there are legitimate users but there are also three attackers, one of which is conducting an APT. It is the actions of this last actor that we want to link together.

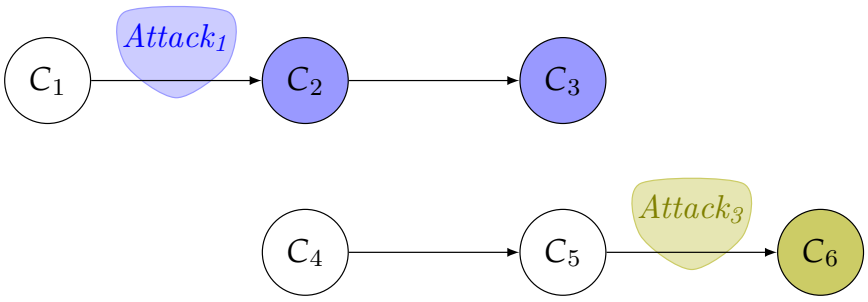
As explained before the aim of this IFT step is to find links between attacks and reconstruct potential attack campaigns. We start with the data from the IDS and extract the flows of information. From the flows of information, we create links between the containers and propagate the tags associated with attacks. For example, between the first attack of the scenario and the second attack, we find the containers shown in Figure 3.5 on page 52. In the figure, the tags are represented by colours. Since the first attack is a shellshock attack against the web server, we see that the `httpd` process is tagged with the tag of the attack (blue in the figure). A number of processes are then executed and a small Remote Access Tool is spawned, disguising itself as `sshd`, a legitimate and usually installed service. This process is used to check the system and create a reverse shell, which is the second attack. From then on, the tag of the second attack is propagated (red in the figure). This establishes a link between $Attack_1$ and $Attack_2$.


(a) All attacks have been detected and are on the same path.


(b) IFT is resilient even if the attack in the middle is not detected




(c) All attacks have been detected and are not on the same path.




(d) IFT is not resilient in this case.

Figure 3.4: Examples showcasing the resilience of IFT.

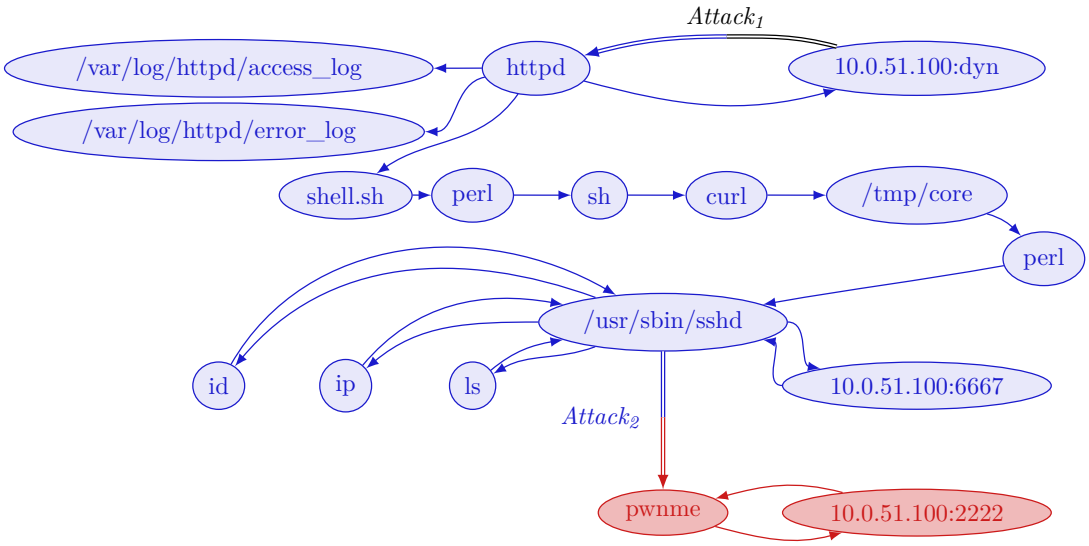
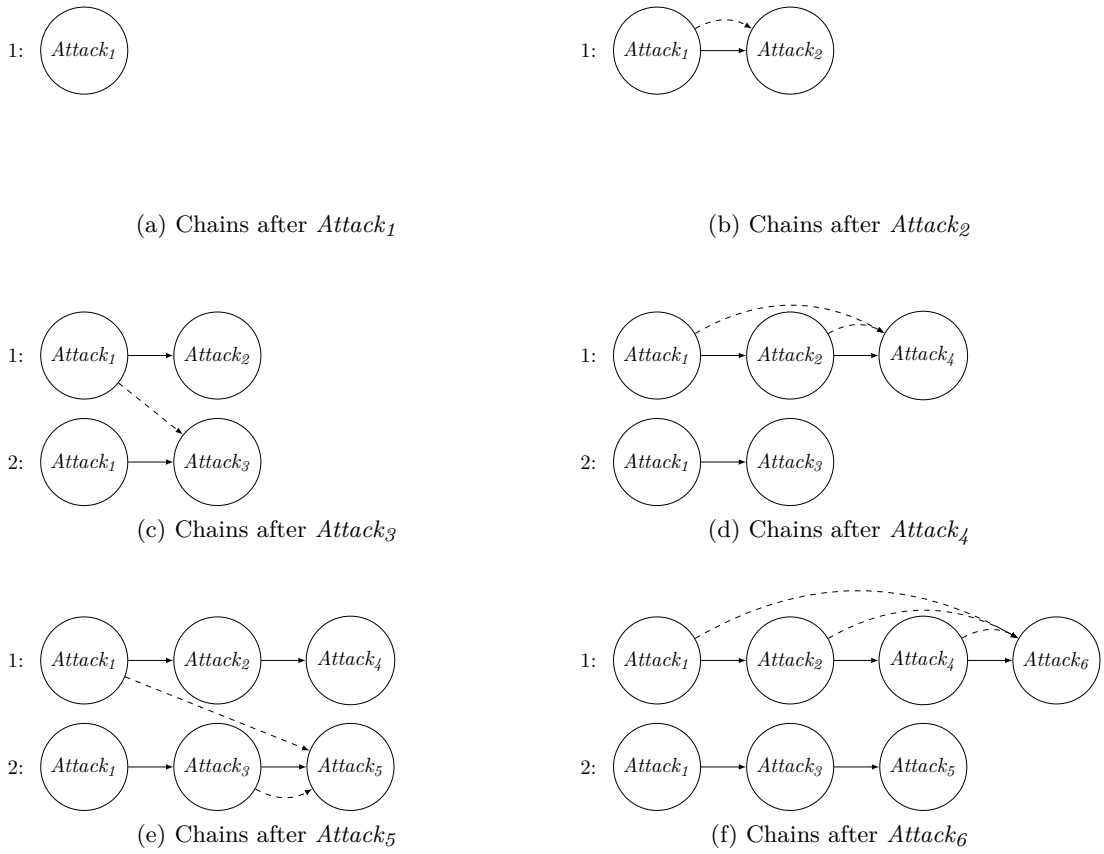



Figure 3.5: Tracking the flows of information from the shellshock attack to the opening of a reverse shell. Arrows represent flows of information and ellipses are containers. A double arrow means that the flow is part of an attack. Each colour represent the tag of an attack, blue being $Attack_1$ and red being $Attack_2$.

Once we discover a link between attacks, we use the AppendAndMergeChains strategy to find the potential attack campaign. Figure 3.6 on the facing page shows the step-by-step evolution of the potential attack campaign whenever we detect an attack. For example, when we detect $Attack_3$, as in 3.6c on the next page, there is a link between $Attack_1$ and $Attack_3$ but none between $Attack_2$ and $Attack_3$. Thus we end up with two potential chains of attacks, one with $Attack_1$ and $Attack_2$ and one with $Attack_1$ and $Attack_3$.

At the end of this phase, we obtain the potential attack campaigns that we must evaluate. In this scenario, we have two potential attack campaigns: "$Attack_1$–$Attack_2$–$Attack_4$–$Attack_6$" and "$Attack_1$–$Attack_3$–$Attack_5$". Since we created the scenario, we know that the first one is a real attack campaign, it is the APT campaign we devised, and that the second one is not a real attack campaign. Instead, it is composed of three independent attacks on the same software, namely `httpd`. This is as expected. IFT found the right attack campaign but also found spurious links. This is why we need the HMM step.




(a) Chains after $Attack_1$

(b) Chains after $Attack_2$

(c) Chains after $Attack_3$

(d) Chains after $Attack_4$

(e) Chains after $Attack_5$

(f) Chains after $Attack_6$

Figure 3.6: Potential chains of attacks obtained through Information Flow Tracking. Dashed arrows show the links found by IFT, and full arrows the link retained by the AppendAndMergeChains procedure.

# Chapter 4

# Hidden Markov Models – Assessing link plausibility

---

*In this chapter, we present the second step of our approach used for filtering the false positives of the previous step. We explain how we create the Hidden Markov Model with the phase of an attack campaign as the hidden states and the categories of attacks as the observations. We establish the matrices by leveraging public APT reports and polling experts. We then design a score using a modified Viterbi algorithm and the probability p that an attack is not detected in order to take into account those potentially missing observations when ranking the potential attack campaigns found by the previous step. By the end of this chapter, we will have a list of potential attack campaigns ranked from most probably an APT to least probably an APT.*

---

## 4.1 Method

### 4.1.1 Concept presentation

A Hidden Markov Model (HMM) is a two-level stochastic process. The first level is a Hidden Markov chain that cannot be directly measured. A Markov chain is a stochastic process following the Markov property, meaning that the process is memoryless, i.e. the future state $s_{t+1}$ of the process, conditional on the current and past states, depends only on the current state $s_t$:

$$\mathbb{P}(s_{t+1} = S_{i_{t+1}} | s_t = S_{i_t}, s_{t-1} = S_{i_{t-1}}, \ldots, s_0 = S_{i_0}) = \mathbb{P}(s_{t+1} = S_{i_{t+1}} | s_t = S_{i_t}).$$

While the Hidden Markov chain cannot be directly measured, its states do generate observations which can be measured. These observations form the second level of the HMM, and the observation at a given time depends only on the state at that time:

$$\mathbb{P}(o_t = O_t | s_t = S_t, s_{t-1} = S_{t-1}, \ldots, s_0 = S_0, o_{t-1} = O_{t-1}, \ldots, o_0 = O_0) = \mathbb{P}(o_t = O_t | s_t = S_t).$$

Thus, a HMM is defined by:

1. The $N$ states of the system, denoted as

   $$S = \{S_1, S_2, \ldots, S_N\}.$$

   This is the Hidden Markov chain. The $L$ states of a given chain of length $L$ of a HMM are written as $s = s_1, s_2, \ldots, s_L$.

2. The $M$ observations of the system, denoted as

   $$O = \{O_1, O_2, \ldots, O_M\}.$$

   These are generated by the states of the Markov chain. The $L$ observations of a given chain of length $L$ of a HMM are written as $o = o_1, o_2, \ldots, o_L$.

3. The state transition probability matrix $A = [a_{ij}]$, where

   $$\forall t,\ \mathbb{P}(s_{t+1} = S_j | s_t = S_i) = a_{ij} \qquad \text{for } i, j \in [\![1, N]\!]. \tag{4.1}$$

   $A$ is a $N \times N$ matrix, and $\forall\ i,\ \sum_j a_{ij} = 1$.

4. The observation probability matrix $B = [b_{ij}]$, where

   $$\forall t,\ \mathbb{P}(o_t = O_j | s_t = S_i) = b_{ij} \qquad \text{for } i \in [\![1, N]\!] \text{ and } j \in [\![1, M]\!].$$

   $B$ is a $N \times M$ matrix, and $\forall\ i,\ \sum_j b_{ij} = 1$.

5. The initial probability vector $\pi = [\pi_i]$, where

   $$\mathbb{P}(s_1 = S_i) = \pi_i \qquad \text{for } i \in [\![1, N]\!]. \tag{4.2}$$

   $\pi$ is a vector of length $N$ and $\sum_i \pi_i = 1$.

We will denote an HMM as $\lambda = (A, B, \pi)$.

A concrete example of an HMM application is described in [Stamp 2004]. The idea is that we want to know whether past years were hot or cold. However, the target years predate temperature records. Hence we do not have access to measurements. Instead,

we use the size of tree rings for those years to try and determine whether they were hot or cold. In hot years, trees will tend to grow more and have larger tree rings and in colder years they grow less and have smaller rings. In this example, the Hidden Markov chain is the sequence of yearly temperature, and we have $N = 2$ states which are $S = \{\text{hot}, \text{cold}\}$. The observations are the tree rings and we have $M = 3$ and $O = \{\text{small}, \text{medium}, \text{large}\}$. The example also specifies the matrices:

$$A = \begin{bmatrix} 0.7 & 0.3 \\ 0.4 & 0.6 \end{bmatrix}, \qquad B = \begin{bmatrix} 0.1 & 0.4 & 0.5 \\ 0.7 & 0.2 & 0.1 \end{bmatrix}, \qquad \pi = \begin{bmatrix} 0.6 & 0.4 \end{bmatrix}.$$

Given an HMM $\lambda$ and a sequence of observations $o$, we can use Baum's forward-backward algorithm [Devijver 1985] to efficiently compute the probability that this sequence of observations was generated by the model:

$$\mathbb{P}(o|\lambda) = \sum_S \mathbb{P}(o|s, \lambda) \cdot P(s|\lambda) = \sum_S \mathbb{P}(o_L|s_L) \cdot \mathbb{P}(s_L|s_{L-1}) \cdots \mathbb{P}(o_1|s_1) \cdot \mathbb{P}(s_1).$$

In addition, we can also compute the most probable sequence of states which generated that sequence of observations as well as its probability in the model using the Viterbi algorithm [Forney 1973]. The Viterbi algorithm is executed by walking forward in the trellis of possible transitions and computing the maximum probability of reaching each state at each time step, taking the observations into account. Once the end of the chain is reached, the trellis is walked backward starting from the highest probability and finding which state in the previous step lead to it and so on until the beginning of the chain is reached.

Using the tree example on the facing page, let us walk through the Viterbi algorithm for the following chain of observations: $o$ = "S", "M", "S", "L". We abbreviate hot as "H", cold as "C", small as "S", medium as "M" and large as "L". The trellis is shown in Figure 4.1 on the next page. From Equation (4.2) on the facing page, we know directly that $\mathbb{P}(s_1 = \text{"H"}) = \pi_{\text{"H"}} = 0.6$ and that $\mathbb{P}(s_1 = \text{"C"}) = 0.4$. We can then compute the probability that the chain is at either state and generates the first observation "S":

$$\mathbb{P}(o_1 = \text{"S"}|s_1 = \text{"H"}) \cdot \mathbb{P}(s_1 = \text{"H"}) = 0.06, \tag{4.3}$$

$$\mathbb{P}(o_1 = \text{"S"}|s_1 = \text{"C"}) \cdot \mathbb{P}(s_1 = \text{"C"}) = 0.28. \tag{4.4}$$

Then, for each state, we use the result of Equation (4.3) and Equation (4.4), multiply by the transition probability and take the maximum of the two. For example, if the second

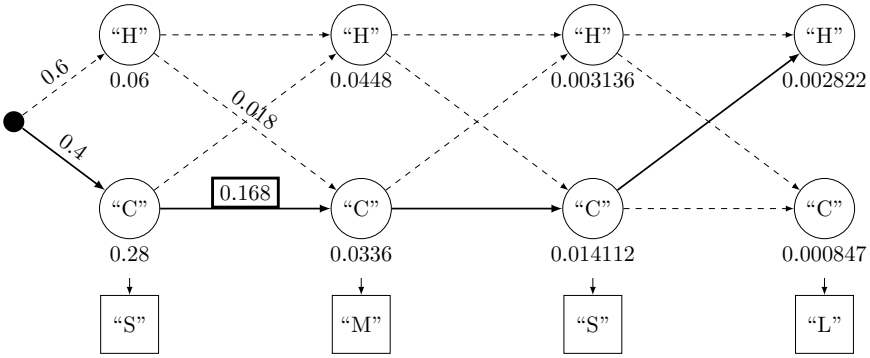



Figure 4.1: Walking the trellis in the Viterbi algorithm for the "temperature of the year and tree ring size" example. We abbreviate hot as "H", cold as "C", small as "S", medium as "M" and large as "L".

state is "C":

$$\max\left(0.06 \cdot \mathbb{P}(s_2 = \text{"C"} | s_1 = \text{"H"}),\ 0.28 \cdot \mathbb{P}(s_2 = \text{"C"} | s_1 = \text{"C"})\right) = 0.168.$$

We then multiply this result by the observation probability $0.168 \cdot \mathbb{P}(o_2 = \text{"M"} | s_2 = \text{"C"}) = 0.168 \cdot 0.2 = 0.0336$. We do this for every state at each step and obtain the numbers in Figure 4.1. We then retrace the trellis by selecting the state with the highest probability ("H" in this case) and finding which state at the previous step led to this probability ("C" in this case). We then do the same for that state and so on until we reach the first step. In our case this leads to the finding that the chain of observation "S-M-S-L" was most probably generated by the state sequence "C-C-C-H" and that $\mathbb{P}(o = \text{"S-M-S-L"}, s = \text{"C-C-C-H"} | \lambda) = 0.002822$.

### 4.1.2 Definitions

As we have seen in the previous section, using a Hidden Markov Model, we can determine the most probable state sequence based only on the measured observations. To apply this method to the problem of recognising attack campaigns, we must first define what are the states and what are the observations. If we go back to the problem definition, we want to find which attacks are part of the same attack campaign. We know the different phases in the typical life cycle of an attack campaign is thanks to Section 1.2 on page 3. And using IFT, we can obtain links between attacks; however, some of these links are spurious so we need to identify the real links. Moreover, the IDS detects attacks and characterise them, but each of those characteristics does not map to only one phase of an attack campaign. This is where we aim to use an HMM. We want to know the state of the system, i.e. in which phase of an attack campaign the attack is. However,

we do not have direct access to this information. Instead, we know the characteristics of each attack. By creating a HMM, we can use it to infer the phase of an attack based on its characteristics and the evolution of the whole chain. Hence, we must create a HMM with the phase of an attack campaign as its hidden states and the characteristics of the attacks as observations. In summary, there are $N = 5$ states $S = \{$ "reconnaissance", "compromission", "establish presence", "privilege escalation", "mission completion" $\}$ and there are $M = 7$ observations $O = \{$ "scan", "arbitrary execution", "credential theft", "application exploit", "backdoor", "remote access tool", "data exfiltration" $\}$. There is a last condition required in order to create a HMM: the underlying process must respect the Markov property. In an attack campaign, the attackers have a goal and are trying to reach it. Hence, in each phase, depending on its results, we can assume that the attackers will either be able to move forward with their plan or will have to backtrack and find another angle of attack. This decision will only depend on the results of the current step; whatever steps where needed to reach this step are irrelevant. Thus, we can assume that the steps of an attack campaign do respect the Markov property.

Once we have defined the states and the observations, the next step is the determination of the matrices. The first source of information for the creation of the matrices are the APT reports written by security companies. While these do a great job of analysing the tools used in each campaign, they do not detail the evolution of the attack campaigns. Hence, we use them to create the observation probability matrix, but we require more information for the transition probability matrix and the initial probability vector. For those matrices, we decided to use expert knowledge. We created a website, detailed in Section 5.3 on page 83, which displays randomly generated attack scenarios. Experts were asked to rank scenarios as "Strongly APT", "Weakly APT" and "Not APT". In addition, experts could remove some steps in the scenario before evaluating it in order to make it more APT-like. This was necessary because most randomly generated scenarios do not look like APT. Using these evaluations, we were able to create the transition matrix and the initial probability vector.

### 4.1.3 The base length-independent score

We are using the HMM to filter the potential attack campaigns found by IFT. This means that we aim to rank the different potential campaigns from most probably an APT to least probably an APT. In particular, we must be able to compare campaigns of different lengths. However, most scores used for HMM are used to do model selection, i.e. find the best matching model for a given chain. This means they are used to compare models and not chains. In effect, scores such as the Akaike Information Criterion (AIC) [Akaike 1974] or the Bayesian Information Criterion (BIC) [Schwarz 1978],

$$L \cdot \ln\left(\min_{ij}(b_{ij})\right) + (L-1) \cdot \ln\left(\min_{ij}(a_{ij})\right) + \ln\left(\min_{i}(\pi_i)\right) \leq \mathcal{L} \leq L \cdot \ln\left(\max_{ij}(b_{ij})\right) + (L-1) \cdot \ln\left(\max_{ij}(a_{ij})\right) + \ln\left(\max_{i}(\pi_i)\right),$$

$$2 \cdot L \cdot \ln\left(\min_{ij}(b_{ij}, a_{ij}, \pi_i)\right) \leq \mathcal{L} \leq 2 \cdot L \cdot \ln\left(\max_{ij}(b_{ij}, a_{ij}, \pi_i)\right), \tag{4.5}$$

$$2 \cdot \ln\left(\min_{ij}(b_{ij}, a_{ij}, \pi_i)\right) \leq \frac{\mathcal{L}}{L} \leq 2 \cdot \ln\left(\max_{ij}(b_{ij}, a_{ij}, \pi_i)\right). \tag{4.6}$$

---

try to balance the accuracy of the model (measured through the log-likelihood $\mathcal{L}$) with its complexity (measured through the number of parameters of the model), the aim being to find a model that is accurate enough but not too complex. The BIC, furthermore, aims to limit the amount of data required for model creation. Since this is not what we require, we must introduce a new score adapted to our APT detection problem. In particular, we must be careful that this new score is not sensitive to the length $L$ of the Markov chain since the log-likelihood, the basis for AIC and BIC, is sensitive to it and we want to be able to compare chains of different lengths.

The log-likelihood $\mathcal{L}$ of a given chain is defined as $\mathcal{L} = \ln(\mathbb{P}(o, s|\lambda))$, where $s$ is obtained using the Viterbi algorithm, meaning it is the most probable chain of states having generated these observations. This definition of the log-likelihood, instead of the more usual $\mathcal{L} = \ln(\mathbb{P}(o|\lambda))$, is specific to the use of the Viterbi algorithm. Since we have access to the most probable state sequence, we can compute the log-likelihood of the sequence of observation having been generated by the sequence of state in the given model. See [Forney 1973] for details on why the state sequence is a given in the probability in this case. We then have:

$$\begin{aligned}\mathcal{L} &= \ln(\mathbb{P}(o, s|\lambda)) = \ln(\mathbb{P}(o_L|s_L) \cdot \mathbb{P}(s_L|s_{L-1}) \cdots \mathbb{P}(o_1|s_1) \cdot \mathbb{P}(s_1)) \\ &= \sum_{l=1}^{L} \ln(\mathbb{P}(o_l|s_l)) + \sum_{l=2}^{L} \ln(\mathbb{P}(s_l|s_{l-1})) + \ln(\mathbb{P}(s_1)).\end{aligned}$$

The log-likelihood is, thus, a sum of negative terms, and the number of terms increases with the length $L$ of the chain. We can easily find an upper and a lower bound on the log-likelihood (see Equation (4.5)). These bounds are dependent on $L$ but the ratio $\frac{\mathcal{L}}{L}$ is bounded by values independent of $L$ (see Equation (4.6)). In keeping with the spirit of

AIC and BIC we define our score as:

$$\mathcal{S} = -\frac{\mathcal{L}}{L}.$$

### 4.1.4 Adapting the score for missing observations: a naïve approach

Now that we have a base score, we also have to take into account the fact that we may miss observations. As we have seen, APTs are executed by skilled actors, this means some steps of the attack will probably escape detection, and the score must account for this possibility. We do this by modifying the Viterbi algorithm.

Due to the Chapman-Kolmogorov equations for homogeneous HMMs, the transition probability of going from state $S_i$ to state $S_j$ in two steps is

$$\mathbb{P}(s_{t+2} = S_j | s_t = S_i) = \sum_k a_{ik} \cdot a_{kj} = A^2(i, j). \tag{4.7}$$

One can then generalise Equation (4.7) and show that the transition matrix of taking $k$ steps when going from one observation to the next is $A^k$. Similarly, the initial probability vector when reaching the first observation in $k$ steps is $\pi \cdot A^{k-1}$. Thus, we denote the HMM where we take $k$ steps from one observation to the next as $\lambda^{(k)} = (A^{(k)}, B, \pi^{(k)})$ where $A^{(k)} = A^k = [a_{ij}^{(k)}]$ and $\pi^{(k)} = \pi \cdot A^{k-1}$. Note that $A^{(k)}$ denotes the transition matrix for the case where there are $k$ steps between one observation and the next, and $A^k$ is the standard matrix power notation.

The idea is to modify the Viterbi algorithm to include the additional transitions described by $A^{(k)}$, i.e. transitions where $k-1$ observations are missed in a row. We denote the state $S_i$ reached in $k$ steps as $S_i^{(k)}$. Thus we have:

$$\mathbb{P}(s_{t+1} = S_j^{(k)} | s_t = S_i^{(l)}) = \mathbb{P}(s_{t+1} = S_j^{(k)} | s_t = S_i) = a_{ij}^{(k)}.$$

Note, in particular, that this does not depend on the value of $l$ which means it does not depend on how many steps it takes to arrive on $S_i$ but only on how many steps are taken going from $S_i$ to $S_j$. If we simply rewrite the Viterbi trellis by adding the states reachable in up to $K$ steps, then we duplicate each state $S_i$ $K$ times in the trellis. Hence, we have to divide each transition probability by $K$ to compensate the effect of this duplication.

This means replacing Equation (4.1) to Equation (4.2) on page 56 with, respectively:

$$\text{for } i,j \in [\![1,N]\!], \quad l \in [\![1,M]\!], \quad k \in [\![1,K]\!],$$

$$\forall t, \qquad \mathbb{P}(s_{t+1} = S_j^{(k)} | s_t = S_i) = \frac{a_{ij}^{(k)}}{K}, \tag{4.8}$$

$$\forall t, \qquad \mathbb{P}(o_t = O_l | s_t = S_i^{(k)}) = b_{il}, \tag{4.9}$$

$$\mathbb{P}(s_1 = S_i^{(k)}) = \frac{\pi_i^{(k)}}{K}. \tag{4.10}$$

We can use Equation (4.8) to Equation (4.10) on this page in the new trellis to compute the Viterbi path and its associated log-likelihood. We note the log-likelihood computed with the Viterbi algorithm allowing for up to $K$ steps from one observation to the next as $\mathcal{L}^{(K)}$, and the score computed from this log-likelihood is:

$$\mathcal{S}^{(K)} = -\frac{\mathcal{L}^{(K)}}{L}.$$

If we adapt the tree example on page 56 with $K = 2$, we have the following matrices:

$$A^{(1)} = A = \begin{bmatrix} 0.7 & 0.3 \\ 0.4 & 0.6 \end{bmatrix}, \quad A^{(2)} = A^2 = \begin{bmatrix} 0.61 & 0.39 \\ 0.52 & 0.48 \end{bmatrix},$$

$$B = \begin{bmatrix} 0.1 & 0.4 & 0.5 \\ 0.7 & 0.2 & 0.1 \end{bmatrix},$$

$$\pi^{(1)} = \pi = \begin{bmatrix} 0.6 & 0.4 \end{bmatrix}, \quad \pi^{(2)} = \pi \cdot A = \begin{bmatrix} 0.58 & 0.42 \end{bmatrix}.$$

Walking the tree is done as in the previous case but using the new equations. From Equation (4.10), we have $\mathbb{P}(s_1 = \text{“H”}^{(1)}) = \frac{\pi_{\text{“H”}}^{(1)}}{K} = \frac{0.6}{2} = 0.3$, $\mathbb{P}(s_1 = \text{“C”}^{(1)}) = 0.2$, $\mathbb{P}(s_1 = \text{“H”}^{(2)}) = 0.29$ and $\mathbb{P}(s_1 = \text{“C”}^{(2)}) = 0.21$. We can then compute the probability for each state that the chain generates the observation “S”:

$$\mathbb{P}(o_1 = \text{“S”} | s_1 = \text{“H”}) \cdot \mathbb{P}(s_1 = \text{“H”}^{(1)}) = 0.015, \tag{4.11}$$

$$\mathbb{P}(o_1 = \text{“S”} | s_1 = \text{“C”}) \cdot \mathbb{P}(s_1 = \text{“C”}^{(1)}) = 0.07, \tag{4.12}$$

$$\mathbb{P}(o_1 = \text{“S”} | s_1 = \text{“H”}) \cdot \mathbb{P}(s_1 = \text{“H”}^{(2)}) = 0.0145, \tag{4.13}$$

$$\mathbb{P}(o_1 = \text{“S”} | s_1 = \text{“C”}) \cdot \mathbb{P}(s_1 = \text{“C”}^{(2)}) = 0.0735. \tag{4.14}$$

Then, for each state, we take Equation (4.11) to Equation (4.14) on the current page, multiply that by the transition probability from Equation (4.8) on the facing page and

take the maximum:

$$\begin{aligned}\max(&0.015 \cdot \mathbb{P}(s_2 = \text{“H”}^{(2)} | s_1 = \text{“H”}), 0.07 \cdot \mathbb{P}(s_2 = \text{“H”}^{(2)} | s_1 = \text{“C”}),\\ &0.0145 \cdot \mathbb{P}(s_2 = \text{“H”}^{(2)} | s_1 = \text{“H”}), 0.0735 \cdot \mathbb{P}(s_2 = \text{“H”}^{(2)} | s_1 = \text{“C”}))\\ =&0.03822.\end{aligned}$$

We then multiply this result with the observation probability from Equation (4.9) on the preceding page and obtain $0.01911 \cdot \mathbb{P}(o_2 = \text{“M”} | s_2 = \text{“H”}^{(2)}) = 0.01911 \cdot \frac{0.4}{2} = 0.003822$. We do this for each state at each step and once we reach the end of the trellis, we start at the state with the highest probability and walk back to find which succession of states leads to this probability. In this case, we find that the chain “S-M-S-L” was most probably generated by the sequence “$C^{(2)}$-$H^{(2)}$-$C^{(2)}$-$H^{(2)}$” which actually means that the observation sequence is “X-S-X-M-X-S-X-L” and the state sequence is “Y-C-Y-H-Y-C-Y-H”, where “X” means that the observation was not observed and ‘Y” that the state is unknown because the associated observation was not observed. It is interesting to note that if we compare this second result to the initial one of “C-C-C-H”, the only known state that changed is the second one, going from “C” to “H”. The associated observation is “M” which is the most ambiguous one; small rings are almost always due to a cold year and large rings to hot years, but medium rings could be either. Seeing only the state associated with this ambiguous observation change when the information is slightly modified, here simply making allowances for incomplete information, is remarkable.

#### Adjusting the value of $K$

Now that we have defined a score, we aim to find an optimal value of $K$ for our application. There is no universal correct value for $K$ because it represents a trade-off between enabling the possibility of missing observations and being able to discriminate matching and non-matching chains. The larger $K$ is the more observations we can consider missing in a row, but the harder it gets to discriminate because the score of every chain gets better. It is important to note that the value of $K$ is capped by $N$, the number of states. This can be shown easily by demonstrating that if a state appears twice between observations, the chain between the two appearances can be removed, including the second appearance, to improve the transition probability. Let us consider a HMM with $N = 4$ states $S = \{A, B, C, D\}$. We take $K = 5$ and want to find the best transition from $A$ to $D$. One possibility is $A - B - C - B - D$ and the associated transition probability is $\mathbb{P}(A) \cdot \mathbb{P}(B) \cdot \mathbb{P}(C) \cdot \mathbb{P}(B) \cdot \mathbb{P}(D)$. Since those are probabilities, we have

$$0 \leq \mathbb{P}(C) \cdot \mathbb{P}(B) \leq 1.$$

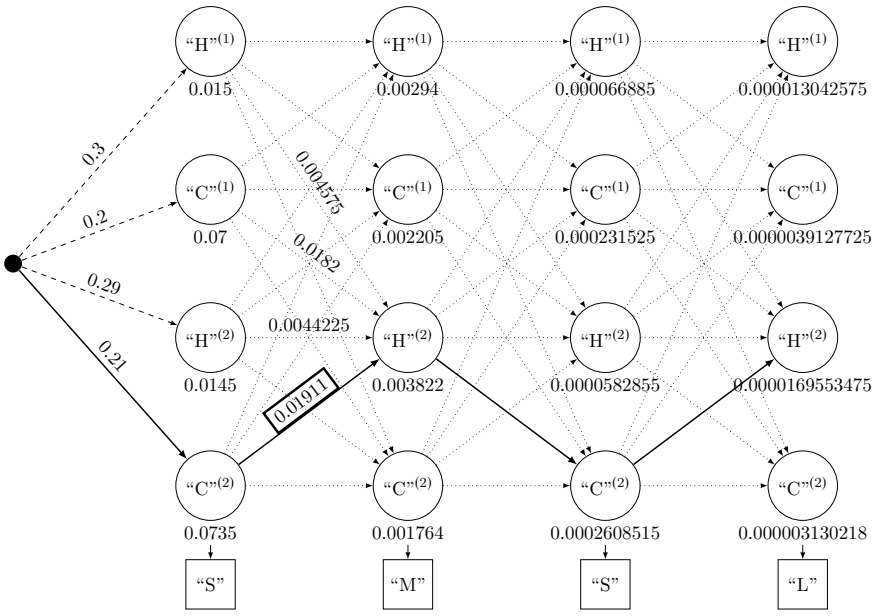



Figure 4.2: Trellis of the Viterbi algorithm with up to $K = 2$ steps from one observation to the next, for the "temperature of the year and tree ring size" example. We abbreviate hot as "H", cold as "C", small as "S", medium as "M" and large as "L".

which in turns means that we have

$$\mathbb{P}(A) \cdot \mathbb{P}(B) \cdot \mathbb{P}(C) \cdot \mathbb{P}(B) \cdot \mathbb{P}(D) \ \leq \ \mathbb{P}(A) \cdot \mathbb{P}(B) \cdot \mathbb{P}(D).$$

This means that whatever the transition probabilities between the states, $A - B - D$ has a better transition probability than $A - B - C - B - D$. This explains why each state may only appear once between each observation, including the observed states, which caps the value of $K$ by the value of $N$.

The test is meant to show that for actual APT chains, the score taking into account missing observations can be a good approximation of the score of the original chain when the value of $K$ is chosen properly. More concretely, we use the chains labeled as "Strong APT" and "Weak APT" (the full dataset is explained in detail in Section 4.2 on page 68). We remove observations randomly and then compute the score for $K$ from 1 to 3. In order to be able to directly compare the scores computed with different values of $K$, we need to adjust them a little. Looking at Equation (4.6) on page 60, we see that our score is bounded by values of the form:

$$\ln\left(\min_{ij}(b_{ij})\right) + \ln\left(\min_{ij}(\frac{a_{ij}^{(k)}}{K}, \frac{\pi_i^{(k)}}{K})\right) = \ln\left(\min_{ij}(b_{ij})\right) + \ln\left(\min_{ij}(a_{ij}^{(k)}, \pi_i^{(k)})\right) - \ln(K).$$

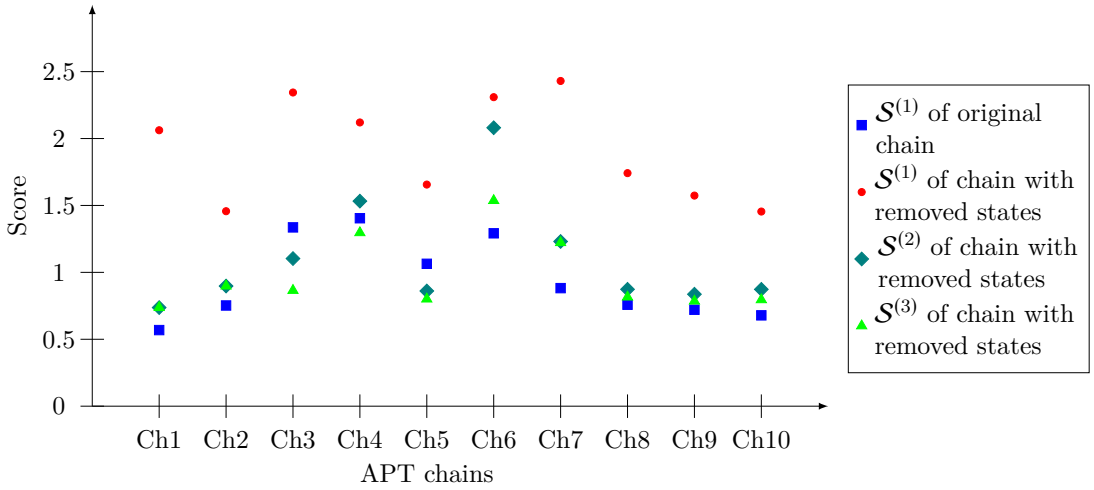



Figure 4.3: The adjusted score increases when we remove states and then comes back closer to the original value as we increase $K$.

This shows that if we want to compare the scores with different values of $K$, we must remove $\ln(K)$ from the score. The adjusted scores are shown in Figure 4.3. They show that when we remove random observations and compute the score $\mathcal{S}^{(1)}$ then the score is worse. However, as the value of $K$ increases, we see that the score comes back to about the same value as the original score.

This method of choosing $K$ is completely subjective, so we tried to apply a more rigorous method. We chose the elbow method. The elbow method consists plotting a metric to optimize against the parameter to choose. We then identify the point on the plot where an increase of the parameter goes from having a large influence on the metric to having a small influence. We must choose a metric for qualifying the quality of the score. For this we use an automatic clustering technique, k-means. Since we want to split the chains of attacks into two classes, "APT" and "Not APT", we do a k-means clustering with two cluster for various values of $K$ and plot the percentage of variance explained by the clustering. A larger percentage is better, so this is the metric we want to optimise. We apply the elbow method on that plot. The result of the k-means clustering on the labeled chains with random states removed is shown in Figure 4.4 on the next page and the plot used for the elbow method is in Figure 4.5 on page 67. We can see immediately, from the first plot, that values of $K > 3$ do not change much to the clustering. In the second plot, we can see that, indeed, the percentage of variance explained is roughly the same whether $K$ is 3, 4 or 5. From this plot, we see that $K = 2$ explains less variance than $K = 3$, so we would choose $K = 3$, which is what we already did previously. It is important to note that we cannot choose the case where $K = 1$ even though it has a higher percentage. This is because $\mathcal{S}^{(1)}$ does not take into account

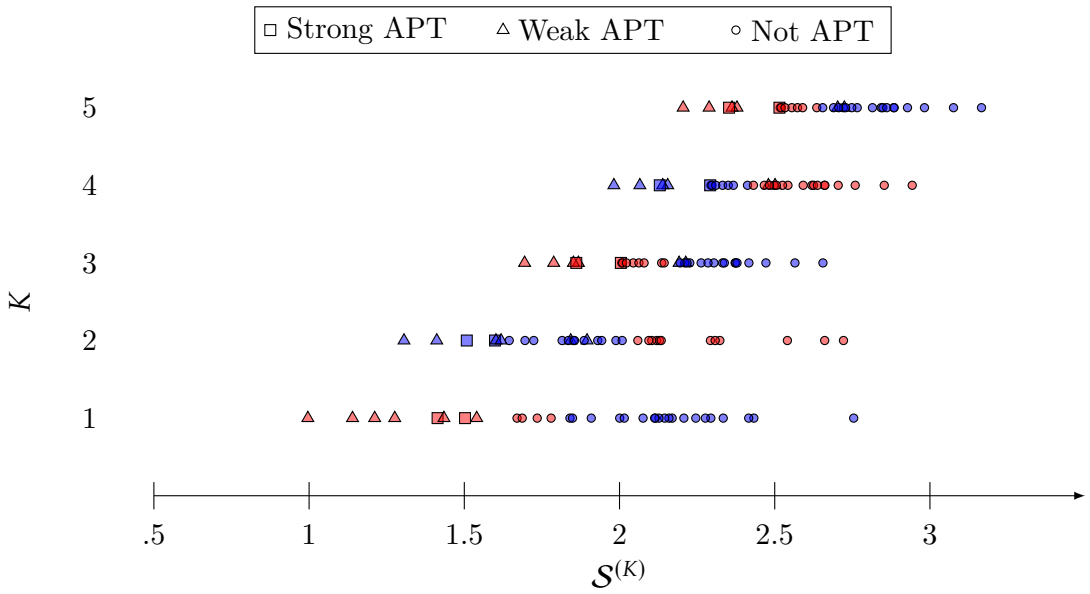



Figure 4.4: k-means clustering with 2 clusters for different values of $K$.

missing observations which is crucial for our use case. In addition, we should also note that we have no ground truth to judge the quality of the clusters. We know the original classification of each chain, but for this plot, random observations have been removed so that the original classification is not representative anymore.

Thus, this score is interesting in evaluating whether a chain of observations fits a given model even if there are potentially missing observations. However, the approach has one big flaw. When we inserted the additional $S_i^{(j)}$ and needed to normalise the transition matrices, we gave equal weight to every matrix by dividing them by $K$. In effect, this means that we decided that the probability of missing $k$ observations in a row is always $\frac{1}{K}$, whatever the value of $k$ as long as $k \leq K$ and 0 if $k > K$; we then chose $K$ in an arbitrarily fashion. The semantics of such a choice are hardly justified, and we would rather have a probability $p$ of missing an observation and use that in our weighing of the matrices. This approach is detailed in the following and will lead to the final score used.

### 4.1.5 A more thoughtful approach

The basis of this approach is the same as before and makes use of the $A^{(k)}$ and $\pi^{(k)}$ matrices. However, this time, instead of weighing each matrix equally, we will use the probability $p$ of missing an observation in the value of the weight. In that case, the probability of going from one measured observation to the next in one step is $1 - p$, the probability of doing so in two steps is $p \cdot (1 - p)$ and the probability of doing so in $k$ steps

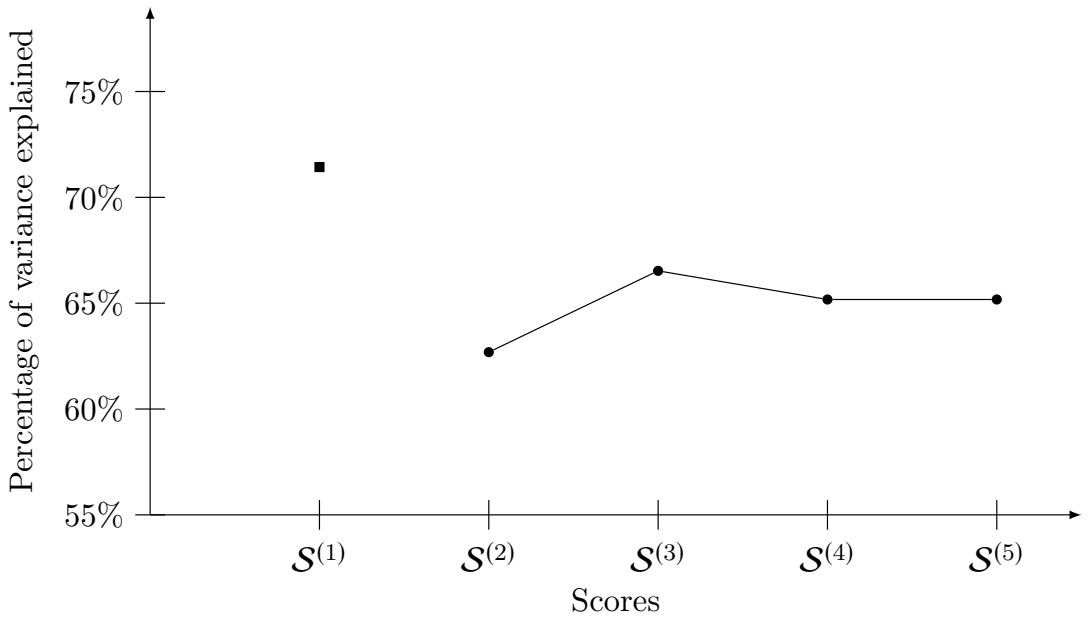



Figure 4.5: Plot of the percentage of explained variance for various values of $K$.

is $p^{k-1} \cdot (1-p)$. We will use these as the basis for the weight of the matrices. However, we still need a $K$ which is the maximum number of steps taken from one measured observation to the next. This is necessary because we cannot compute the matrices for $k \in [\![1, +\infty[\![$. However, in this case, when $k$ becomes larger, $p^{k-1} \cdot (1-p)$ becomes smaller. By setting a limit $K$, we simply decide how good of an approximation we want for the score. Once we have the weights, we do need to normalise them. Equation (4.8) to Equation (4.10) on page 62 become:

$$\begin{aligned}
&\text{for } i, j \in [\![1, N]\!], \quad l \in [\![1, M]\!], \quad k \in [\![1, K]\!], \\
&\qquad \forall t, \qquad \mathbb{P}(s_{t+1} = S_j^{(k)} | s_t = S_i) = a_{ij}^{(k)} \cdot \frac{p^{k-1}}{\sum_{n=1}^{K} p^{n-1}}, \\
&\qquad \forall t, \qquad \mathbb{P}(o_t = O_l | s_t = S_i^{(k)}) = b_{il}, \\
&\qquad\qquad\quad \mathbb{P}(s_1 = S_i^{(k)}) = \pi_i^{(k)} \cdot \frac{p^{k-1}}{\sum_{n=1}^{K} p^{n-1}}.
\end{aligned}$$

We note those matrices as $A_p^{(k)}$ and $\pi_p^k$. We use the tree example on page 56 with this

score, choosing $K = 2$ and $p = 0.8$, and we obtain the following matrices:

$$A_p^{(1)} = A \cdot \frac{1}{1+p} = \begin{bmatrix} 0.389 & 0.167 \\ 0.222 & 0.333 \end{bmatrix},$$
$$A_p^{(2)} = A^2 \cdot \frac{p}{1+p} = \begin{bmatrix} 0.271 & 0.173 \\ 0.231 & 0.213 \end{bmatrix},$$
$$B = \begin{bmatrix} 0.1 & 0.4 & 0.5 \\ 0.7 & 0.2 & 0.1 \end{bmatrix},$$
$$\pi_p^{(1)} = \pi \cdot \frac{1}{1+p} = \begin{bmatrix} 0.333 & 0.222 \end{bmatrix},$$
$$\pi_p^{(2)} = \pi \cdot A \cdot \frac{p}{1+p} = \begin{bmatrix} 0.258 & 0.187 \end{bmatrix}.$$

The trellis obtained is shown in Figure 4.6 on the next page. We find that the chain "S-M-S-L" was most probably generated by the sequence "$C^{(1)}$-$C^{(1)}$-$C^{(1)}$-$H^{(2)}$" which actually means that the observation sequence is "S-M-S-X-L" and the state sequence is "C-C-C-Y-H", where "X" means that the observation was not observed and "Y" that the state is unknown because the associated observation was not observed.

As we can see, the results with this score differs from both the original Viterbi and from the first iteration of our customised score. In terms of states only, this second score matches the original score, however, it does add to the interpretation by showing that there probably is a missing observation between the third and fourth ones. While this example using trees is contrived, especially with $p = 0.8$, in the case of APTs, due to the skills of the attackers, there will be missed observations. By choosing an appropriate value for $p$, we can take them into account.

## 4.2 Evaluation on APT classification

In this case study, we want to apply the model and the score defined above to the problem of APT detection. More specifically, we receive as input a number of sequence of observations and we must determine which of these sequences are more probably APTs. This means that the score must order the sequences from most probably an APT to least probably an APT. Additionally, the model should show the hidden states, including the number of unobserved states, if any.

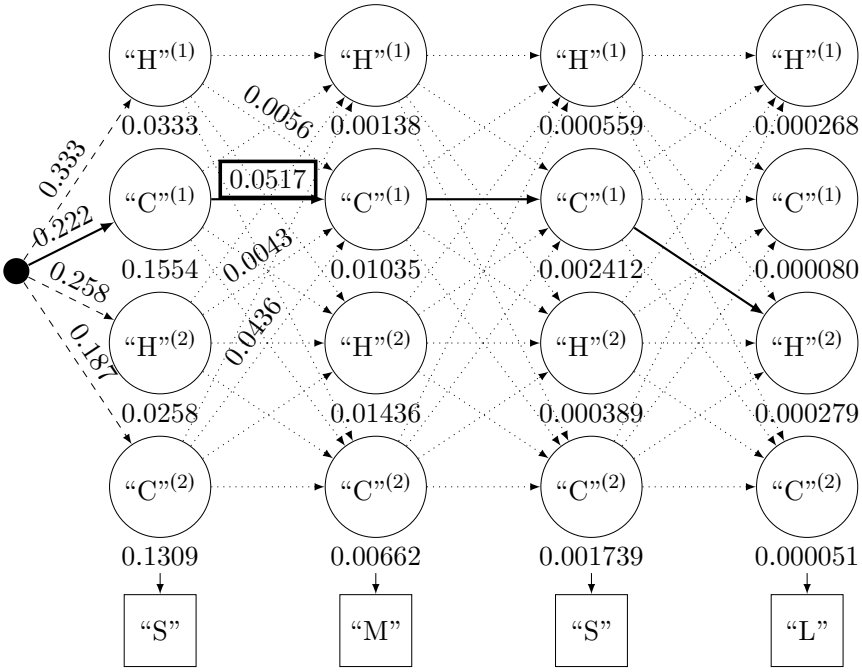



Figure 4.6: Trellis of the Viterbi algorithm with up to $K = 2$ steps from one observation to the next and a probability $p = 0.8$ of missing an observation, for the “temperature of the year and tree ring size” example. We abbreviate hot as “H”, cold as “C”, small as “S”, medium as “M” and large as “L”.

### The proposed model

The first step is to define the list of states $S$ and observations $O$. Our model uses the five phases we identified for APTs in Section 2.1 on page 11: $N = 5$ and $S = \{$ "reconnaissance (R)", "compromission (C)", "establish presence (EP)", "privilege escalation (PE)", "mission completion (MC)"$\}$. For the observations we use the output of a tool which defines seven categories of observations: $M = 7$ and $O = \{$ "scan", "arbitrary execution", "credential theft", "application exploit", "backdoor", "remote access tool", "data exfiltration" $\}$. Once we have the states and observations, we require statistics about the transitions and observations in order to compute the matrices. There are two sources of information that we leverage. First, there are publicly available APT reports. They are useful in knowing which tools are used in which phase, but they are not precise enough in the evolution of the APT. Hence, we use them to create the observation matrix $B$. For the initial probability vector $\pi$ and the transition matrix $A$, we use expert knowledge. We setup a website where experts are shown scenarios created at random. Each scenario can be rated as "strongly an APT", "weakly an APT" or "not an APT". The scenarios are shown as chains of states with an observation presented for each state as an indication. For example, the scenario "R", "C", "PE", "EP" and "MC" was rated as "strongly an APT" while the scenario "MC", "C", "MC", "EP" and "R" was rated as "Not an APT". Thanks to these two sources of information, we obtain the following HMM model $\lambda = (A, B, \pi)$:

$$A = \begin{bmatrix} 0.045 & 0.091 & 0.318 & 0.5 & 0.045 \\ 0.071 & 0.071 & 0.643 & 0.143 & 0.071 \\ 0.045 & 0.182 & 0.045 & 0.682 & 0.045 \\ 0.16 & 0.04 & 0.04 & 0.04 & 0.72 \\ 0.2 & 0.4 & 0.267 & 0.067 & 0.067 \end{bmatrix},$$

$$B = \begin{bmatrix} 0.3 & 0.02 & 0.02 & 0.02 & 0.6 & 0.02 & 0.02 \\ 0.5 & 0.14 & 0.01 & 0.1 & 0.13 & 0.07 & 0.4 \\ 0.01 & 0.3 & 0.01 & 0.3 & 0.05 & 0.03 & 0.3 \\ 0.05 & 0.4 & 0.05 & 0.05 & 0.05 & 0.1 & 0.3 \\ 0.4 & 0.03 & 0.01 & 0.09 & 0.01 & 0.4 & 0.06 \end{bmatrix},$$

$$\pi = \begin{bmatrix} 0.7 & 0.21 & 0.03 & 0.03 & 0.03 \end{bmatrix}.$$

### Experiment 1: Model adequacy checking

The aim of this first experiment is to check that the model differentiates APT chains from non-APT chains. To do so, we compare the scores of various scenarios rated as

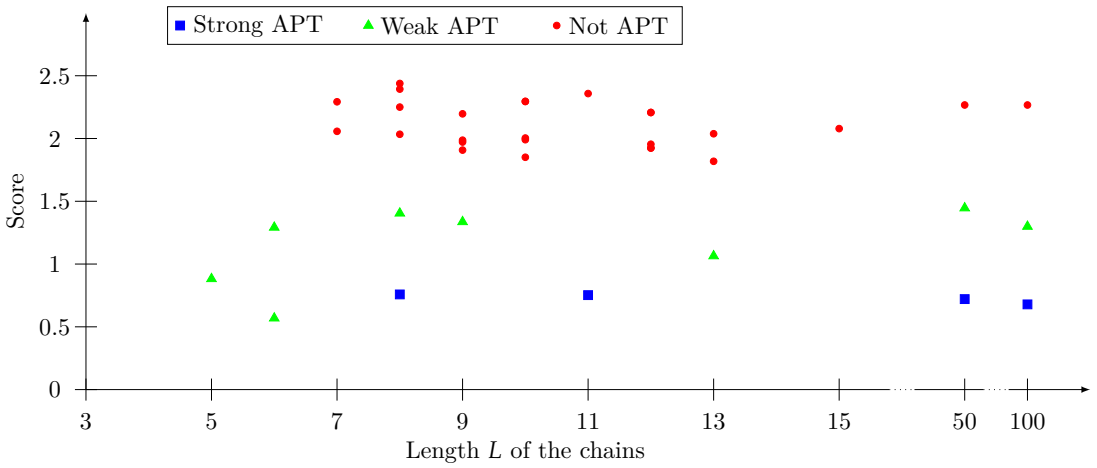



Figure 4.7: The score separates APTs from non-APTs

APT and as non-APT by our expert raters. We can then plot the scores and check that non-APT chains score higher than APT chain. In this first experiment, our score does not take into account possible missing observations. This means that we use the score $\mathcal{S}^{(1)}$. During the poll we explicitly told the expert raters to consider the steps shown as the whole APT, with no missing step, so this matches how the model was created.

The results can be seen in Figure 4.7. The "Strong APT" chains appear in blue, the "Weak APT" chains in green and the "Not APT" ones in red. They show that both the proposed model and the associated score are able to separate APT from non-APT, with all APT chain below 1.5 and all non-APT above that mark.

We also created one chain of each kind with a length of 50 and 100 to check that the score is still consistent when chains are that long. Indeed, we see that the score does not depend on the length of the chain $L$, which was another crucial goal of our score.

#### Experiment 2: Analysing the impact of missing observations

The aim of this second experiment is to check the impact of taking into account possible missing steps on the score and, more importantly, on the rank of each chain evaluated.

We display the same chains as in Figure 4.7 but using $\mathcal{S}^{(3)}$ instead of $\mathcal{S}^{(1)}$ and with different values of $p$ in Figure 4.8 to Figure 4.16 on pages 72–76. As we can see, the higher $p$ is, the more some "Not APT" are on the same level as "APT" chains. This reflects that the higher $p$ is, the higher the probability of missing one or even two consecutive observations is too, which means that our score will consider missing observations more. This is exactly what we wanted: by inserting well chosen and positioned unobserved states in the chain, their likelihood of being APTs is much higher. This means that if we

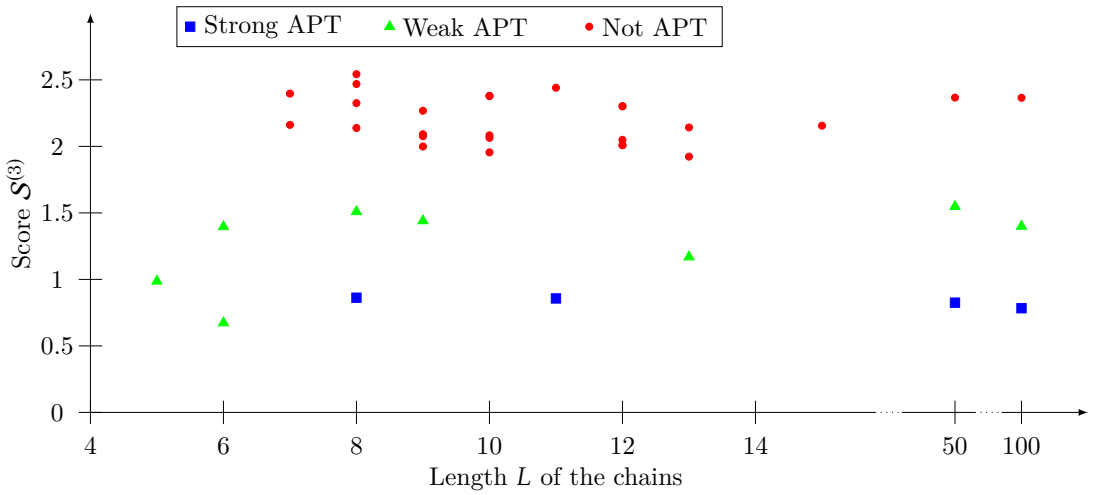



Figure 4.8: $\mathcal{S}^{(3)}$ with $p = 0.1$ of the original scenarios.

have an APT where we did not observe some of the attacks, we can still reconstruct the probable unobserved attacks and evaluate that chain as an APT. For example, if we have a chain for which Viterbi gives us $\mathcal{S}^{(1)} = 1.62$ and the states "reconnaissance"-"establish presence"-"mission completion". From Figure 4.7 on the previous page, this chain is on the "Not APT" side, and there are phases missing in the APT, between "reconnaissance" and "establish presence" for example. Switching to $\mathcal{S}^{(3)}$, if $p = 0.3$, $\mathcal{S}^{(3)} = 1.87$ and the states do not change; from Figure 4.10 on the facing page, the chain is still on the "Not APT" side, and there are still missing phases. However, if $p = 0.7$, $\mathcal{S}^{(3)} = 2.04$, the states become "compromission"-undetected-"privilege escalation"-"mission completion". From Figure 4.14 on page 75, this is now on the "APT" side. This is explained by the fact that if we replace the undetected state by an "establish presence" phase, we now have a very reasonable chain of attacks for an APT.

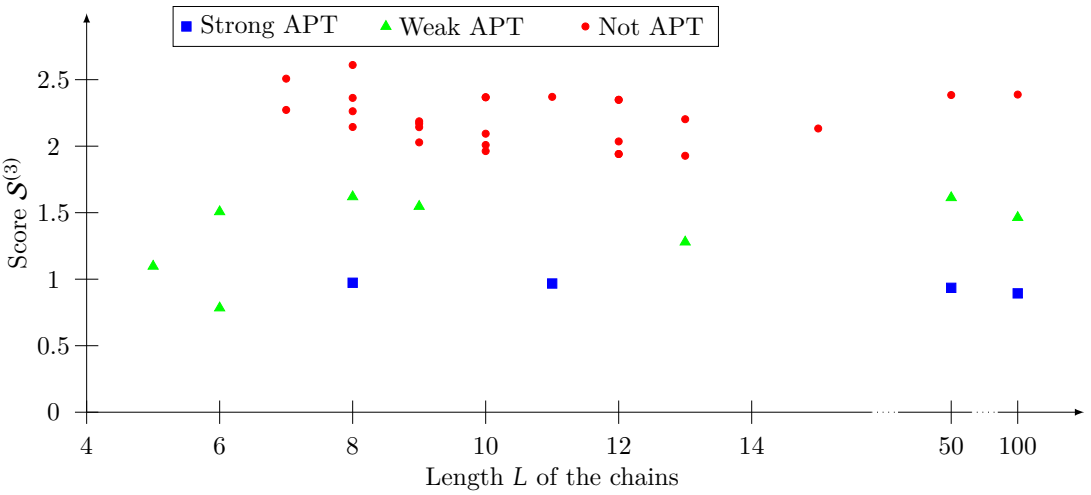



Figure 4.9: $\mathcal{S}^{(3)}$ with $p = 0.2$ of the original scenarios.

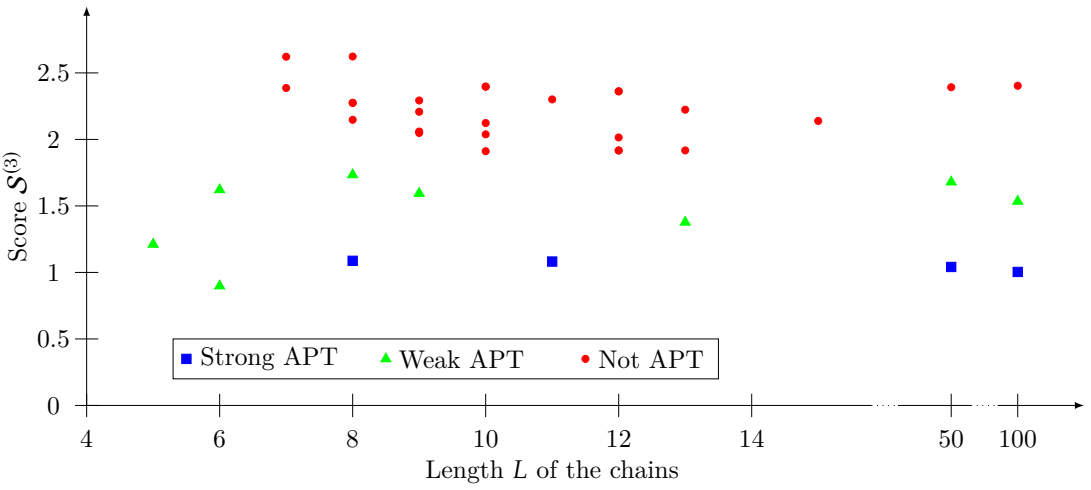



Figure 4.10: $\mathcal{S}^{(3)}$ with $p = 0.3$ of the original scenarios.

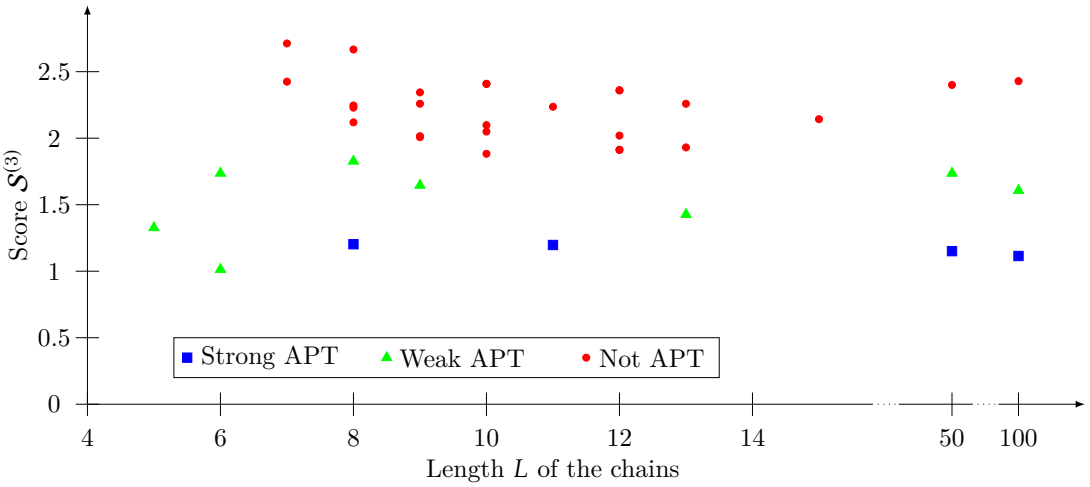



Figure 4.11: $\mathcal{S}^{(3)}$ with $p = 0.4$ of the original scenarios.

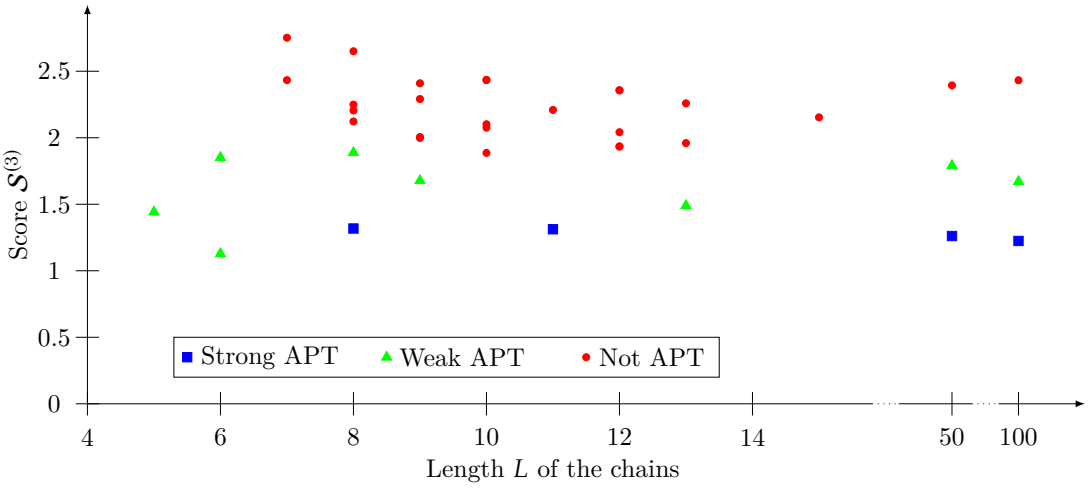



Figure 4.12: $\mathcal{S}^{(3)}$ with $p = 0.5$ of the original scenarios.

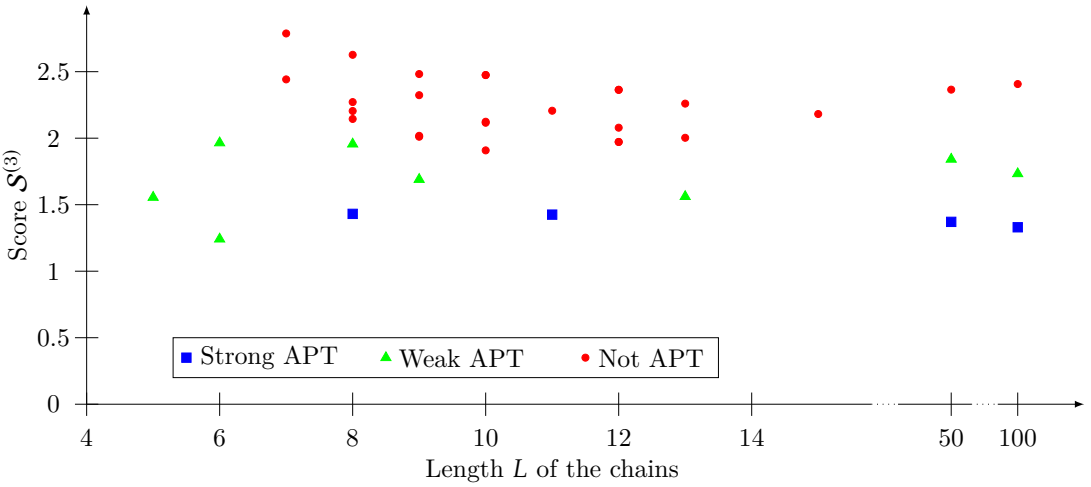



Figure 4.13: $\mathcal{S}^{(3)}$ with $p = 0.6$ of the original scenarios.

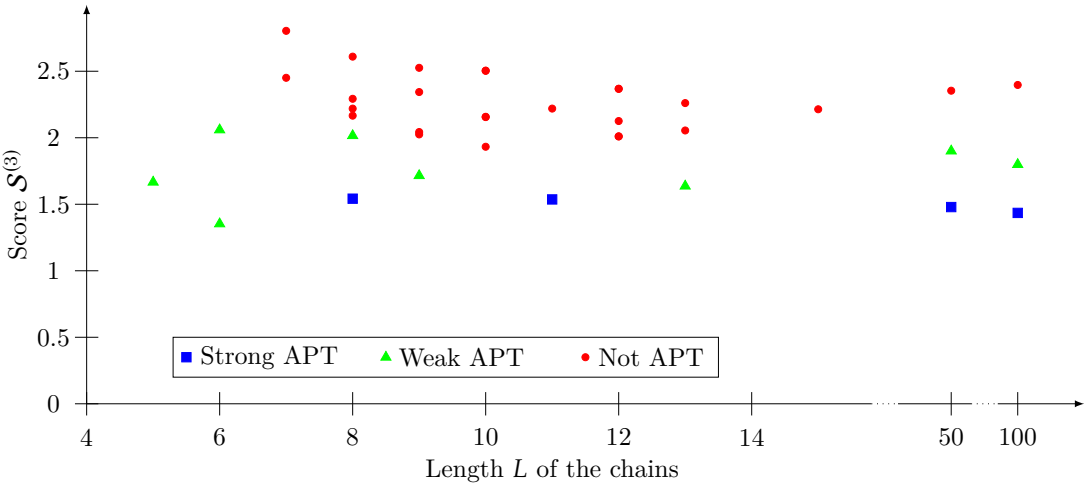



Figure 4.14: $\mathcal{S}^{(3)}$ with $p = 0.7$ of the original scenarios.

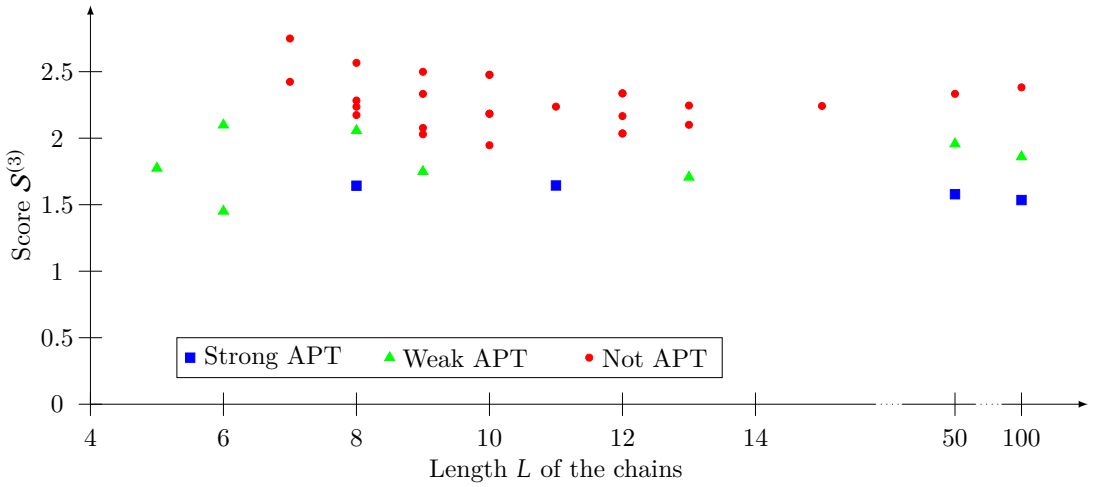



Figure 4.15: $\mathcal{S}^{(3)}$ with $p = 0.8$ of the original scenarios.

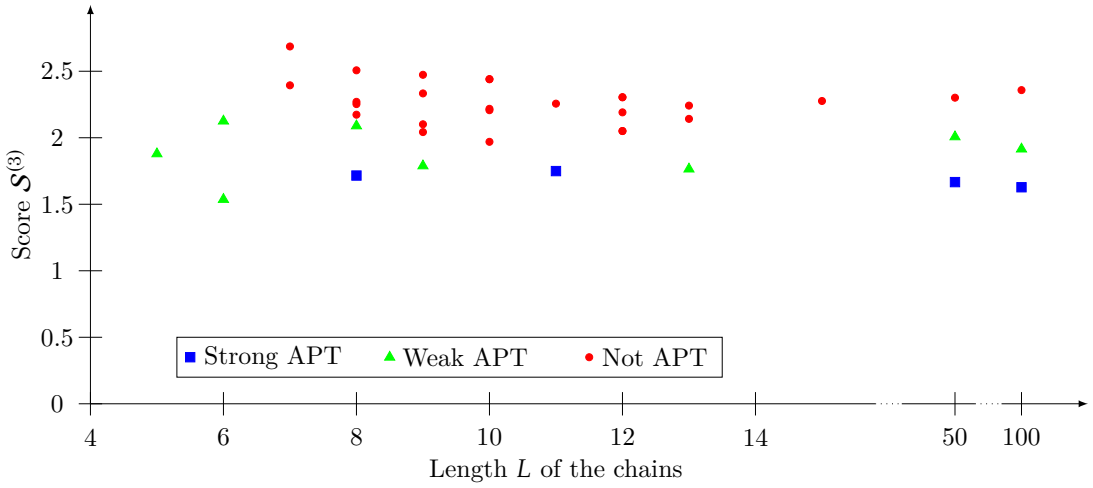



Figure 4.16: $\mathcal{S}^{(3)}$ with $p = 0.9$ of the original scenarios.

# Chapter 5

# Evaluation scenario and tooling

---

*The aim of this chapter is to detail both the tools and the actual scenarios used to evaluate the methods presented in Chapter 3 on page 39 and Chapter 4 on page 55.*

*The first tool is the Intrusion Detection System (IDS) which also collects flows of information. This IDS is the result of three PhD thesis, including this one. In this chapter, we only present how we built the data collection agent, which forms the basis of this IDS and was a requirement for all three PhD thesis.*

*The second tool is used to define and play attack scenarios. There are no public datasets of Advanced Persistent Threat (APT) scenarios, so we had to create our own scenarios, and this tool is meant to reproduce those scenarios exactly and easily.*

*The third and last tool is a website for polling experts. When creating a Hidden Markov Model (HMM) of APTs, we asked experts to rate randomly generated attack scenarios. This knowledge was used in the determination of the initial probabilities and the transition probabilities of the model.*

*Lastly, we will walk through the creation of a scenario from scratch using the tools described in this chapter in order to collect data and evaluate our methods. This is the scenario which will subsequently be used for the evaluation of our methods.*

---

## 5.1 Getting the raw data for the IDS

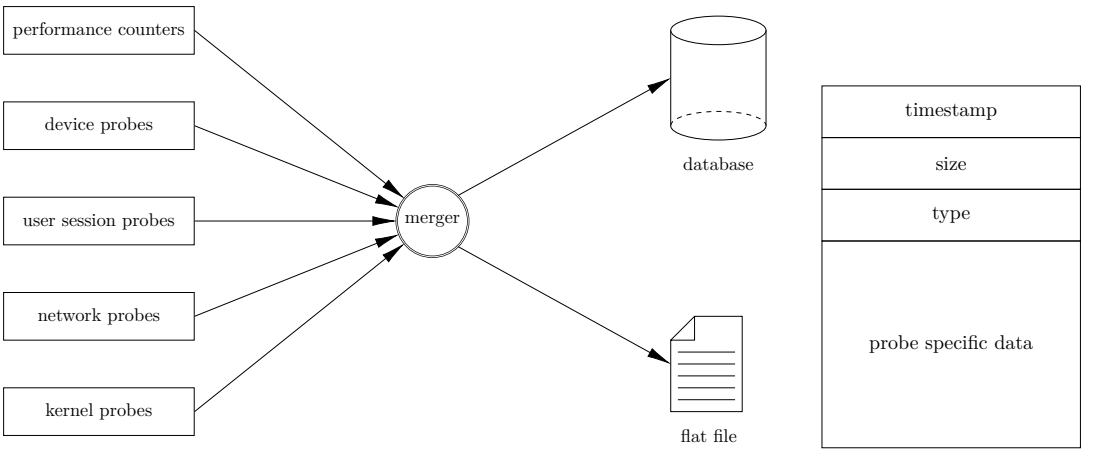
### 5.1.1 Merging data from multiple sources

As mentioned, the IDS collecting flows of information was a collaboration project between three PhD students, including me. The first part that was developed are the probes collecting data about the system to monitor. These data are used by all three projects and so the probes were developed by all three students at the beginning of their PhD.

The initial focus was on building a modular system to collect data from multiple source and make sure that the data remains in chronological order. We knew we wanted to aggregate data from multiple sources but did not know how many. Hence we created a system that can take streams of data from any number of sources as input, makes sure that the ordering is consistent and outputs the merged streams to a single file or database, as shown in 5.1a on the facing page. In order for the system to be able to merge the streams in chronological order, each point of data must contain a timestamp in the same place, whatever its source. At the base of each data point, we inserted metadata containing the timestamp and also the size of the data point and its type, as shown in 5.1b on the next page. The merging system operates as follows. Each probe has a dedicated circular buffer. When a probe collects new data, it writes it to the circular buffer and wakes the merger. The merger then check each buffer for available data, copies the raw data, using the size indicated in the data point itself, in an internal buffer where all the data is in chronological order. For performance reasons, this is implemented as a linked list because we often need to insert data at arbitrary position. Then, regularly, a second thread in the merger will check to see if there is any data point older than a predefined time, 1 second in our case, and dump that data to the file or the database. This delay is needed because some probes could be slower than others and in that case, their data points could arrive after newer data points from other streams. When dumping the data, the merger must, finally, know the structure of the data point it is currently dumping. This is where the type of data, indicated in the data point, is used to convert the complete data structure to a dictionary, which can then be converted to the specified format.

In our case, we used the JSON format for the flat file. This format is particularly adapted to representing dictionaries, especially since each data source has different keys in the dictionary.

One of the criticism of this architecture is that if the merger is too slow compared to the data sources, there is a risk of losing data. This is compounded by the fact that




(a) Architecture of the probes and the data collector

(b) Structure of a data point

Figure 5.1: Data collection

there is a single merger for several probes. In our case, we decided to keep the newer data, i.e. the data streams overwrite old data if they reach the end of the circular buffer. However, in effect, we never lost any data because of this in any of our experiment.

An additional criticism is that the merger may be unnecessary when writing to a database. Once the data is in the database, the data points can be queried in chronological order with the appropriate query. While this is true, in our case, we needed an architecture capable of merging the logs in chronological order and writing them to a flat file. Hence the merger is necessary for our use case. It is also useful if we want to send the data to several outputs at the same time, such as a flat file and a database or two different databases.

### 5.1.2 Collecting the data

Once the merger is in place, we need to write the probes to do the actual data collections. Integrating a new probe with the merger is easy. Each probe runs in its own thread and writes its collected data in its circular buffer. The merger takes care of the rest. The following is a quick overview of each probe and how it collects data. Most probes are self contained, but two require external modules, which we will describe in more detail.

The simplest module is the performance module. Periodically, it collects values about the system. For each process in the system, it measures its CPU and RAM usage as well as the amount of input/output (IO) from/to storage. The process itself is described through its name, its PID and the PID of its parent. The second module follows the

creation, connection and closing of sockets inside the system. The third module collects information about packets sent and received by the system. In particular, it matches the packet with the sending or receiving process on the host. The fourth module collects udev events. udev is the subsystem in Linux which originally managed devices. Its purview has been expanded and now includes events such as the mounting of a new partition and the loading of a kernel module.

The fifth module collects data from PAM, and it is the first one which requires an external module. PAM is the Linux Pluggable Authentication Modules system. PAM handles authentication events and when a new event must be evaluated, this event is run by a number of modules depending on its origin. Each module can decide whether to allow or disallow the authentication, and depending on the order and options associated with each module, PAM will decide whether the authentication is authorised or not. In order to collect authentication attempts as well as their results, we have inserted a PAM module which reports all attempts to the merger module via named UNIX socket.

The sixth module is where we collect data from the kernel. To do so, we created a kernel module using Kprobes [Keniston, Panchamukhi, and Hiramatsu 2005]. Kprobes are a part of the Linux kernel made to audit the kernel. We use it to collect information on a number of system calls. The kernel module communicates with the merger module through a file in the proc virtual file system and uses that to create a shared memory mapping. For performance reasons, there are actually one merger module thread per CPU core and thus one file in proc and one shared memory mapping per core. We use the Kprobes to collect data on process creation and execution, monitor file systems for new files or metadata changes, and all types of sockets. In addition, the kernel module is smart enough to not monitor the process it is talking to, i.e. the merger, which would cause trouble as the merger writing to the flat file would trigger Kprobes events which would then be written and so on, launching a never ending circle of events being logged. Kprobes are originally made for debugging the kernel so we expected potential performance issues. In particular, if there are too many Kprobes being triggered, the system stops processing them and emits a warning. In our tests, we never saw that warning. However, we should investigate the use of eBPF [Schulist, Borkmann, and Starovoitov 2016] instead if we were to rewrite that part of the code. eBPF is originally a packet filter facility and is made with performance in mind instead of debugging which should alleviate the performance concerns, and it should be possible to do everything that we do with Kprobes with eBPF.

An equivalent of all these probes has also been developed for the Windows platform. The software architecture and the data collected are both the same as the Linux probes. The only differences are in how the data are collected. Since we did not participate in

| Measures in $ms$ | Average time | Standard deviation | First quartile | Last quartile |
|---|---|---|---|---|
| Without Akheros | 5.21 | 0.41 | 4.87 | 5.56 |
| With Akheros | 6.28 | 2.77 | 4.14 | 8.00 |
| Difference (%) | 120% | 677% | 85% | 144% |

Table 5.1: Response time of the website

their development, we will not be detailing them. However, we will use them in some of my tests.

### 5.1.3 Performance evaluation

As the merger collects a lot of data from the system, we want to check that its impact does not disrupt the original functions of the system. To do this, we setup a Virtual Machine (VM) with a web server on it and measured the impact of the probes and merger on the responses of the system. This means that we consider the impact as a user of the service and not by measuring usage on the server directly. More precisely, we used the apache benchmark, called `ab`, and instruct it to make a hundred thousand requests to the web server using thirty concurrent requests at a given time. The command was executed from another VM running on the same host. The results can be seen in Table 5.1. On average, the probes and merger add a 20% overhead to the response time. However, these results are for response times on the same host, which means they do not take into account the network latency. Typically the network latency will be in the tens to low hundred of milliseconds on the web. In that case, adding about $1ms$ means less than 5% overhead, which is reasonable.

## 5.2 Reproducible scenarios with Moirai

In order to test the solution we propose for the detection of APTs, we needed a dataset of APT scenarios. However, no such dataset exists, and even if it did, we do not know whether it would contain the necessary data for the Information Flow Tracking. The solution we adopted was to create a tool, Moirai, to make it easy to define and replay scenarios. In addition, this tool makes it easy to share, update and improve those scenarios, thus creating a dataset of APT scenarios that the community can use to check the performance of their solution and compare it with others.

One of the aim of Moirai is to keep the tool as simple as possible and to reuse existing

technologies as much as possible. Thus, Moirai takes a text file, in the `ini` style, as input and plays the scenarios described therein by orchestrating VMs. For the management, and sharing, of VM, Moirai uses Vagrant [HashiCorp 2010]. Vagrant is a tool created to build portable development environments using VMs. Since it is widespread, there is a large library of VMs available in their online repository. This means we can use these VMs as the base of our scenarios. We can then install the necessary software, including the IDS to be tested, on these VMs and play our scenarios. Moirai has a `[Cluster]` section in its configuration file to list of the machines which are then described in corresponding `[machine]` sections. When sharing the configuration file, one can either upload a machine with all the installed software on it to Vagrant's repository, or one can share the name of the base box with the list of software to install on it as well as the associated configuration files. In that second case, which is the case we use, we recommend the use of automation software, such as Ansible [Ansible 2012], to make the installation and configuration of additional software as easy and as faithful to the original as possible.

Once Moirai knows which VMs to fetch from the Vagrant's repository, it sets up the VMs according to the network topology written in the configuration file under the `[machine]` sections. In addition, Moirai will have its own router VM to allow redirecting and spoofing IPs and domain names. This VM will be shared as is on the Vagrant's cloud. This is useful in the case where a malware sample always tries to connect to the same host. We can then transparently redirect any traffic for that host to the VM supposed to emulate it.

Similarly to how Moirai defines the list of VMs in one section and then each VM has its own section, Moirai defines the list of task in the `[Scenario]` section and then each individual task is specified in its `[task]` section. Additionally, the `[Scenario]` section contains scenario wide settings which can, for example, control the maximum run time of the scenario. Each `[task]` section requires a timing and a target. This indicates when and on which machine to execute the task. Task can be composed of up to three components: one or several actions to execute on the target, a list of files to send from the host to the target before the action is executed and a list of files to fetch from the target once the action has been executed.

Listing 1 on page 84 shows an example of a working configuration based on the shellshock scenario available here [Akheros 2016b]. Moirai itself is open source and available here [Akheros 2016a]. It is written in python3 and should be cross-platform. It can control UNIX based machines using ssh and Windows based machines using winrm, even older versions up to Windows XP. If we follow the listing, we can see that it includes two machines (line 2) based on the same box (lines 5 and 9). There are three

tasks (line 13) and the scenario will not last more than 10 minutes (line 14). This is necessary because the `[botmaster]` action (line 19) will run forever otherwise. We can see that the first attack, the `[shellshock]` action, starts at 1 minute (line 24) and that the second attack starts 5 minutes after that (line 30). If one wanted to play this scenario, there is a one time setup to perform. The instructions are on the website where the scenario is first downloaded. The setup requires the user to start the two VM and then use Ansible to configure them. Once the VM are configured, the user should repackage them with Vagrant and modify the lines specifying the base boxes accordingly (lines 5 and 9). Once this initial setup is done, and it only needs to be done once, playing the scenario can be done in a single Moirai command and left unattended. A choice worth noting in this listing is the fact that we decided that for each attack, the attack script will be sent to the attacker right before the action rather than during the setup. This can seem wasteful but it enable tweaking the attacks easily without having to redo the whole setup, and the scripts are small enough that the waste is limited.

## 5.3 Website to poll experts about APTs

In order to create the state transition matrix for the HMM, we needed statistics about the evolution of APTs. Quite a few APTs have been studied extensively and reports have been written about them [Mandiant Intelligence Center 2013; Fireeye Labs 2015; McAfee Labs 2013; Trend Micro 2013]. These reports contain details about which malwares are used in which stages of an APT. However, they do not detail the evolution of each APT campaign, which is why we decided to poll security experts who had experience with APTs on what APTs typically look like. To do this, we built a web application [Brogi 2016], shown in Figure 5.2 on page 86, which displays attack scenarios. These scenarios are generated at random. Security experts can then look at the scenarios and decide whether it looks like an APT by rating it as “not an APT”, “weakly an APT” and “strongly an APT”. For example, looking at 5.2a on page 86, we can see a scenario with one step establishing presence on a server, the attacker then moving to another server (in what is called a lateral movement), four more steps, another server, another four steps and a last server and a last step. The expert seeing this scenario has the option to label it “Not at all”, “Maybe” or “Yes” or to “SKIP” to another scenario. In addition, experts have the option of removing a few steps, such as the struck out steps 3 and 6 in 5.2b on page 86. This allows them to transform a scenario that is definitely not an APT to one that is. This is necessary because randomly generated scenarios are not very likely to look like APTs and this option means that experts needed to look at fewer scenarios before labelling one as an APT. Overall, this reduced the time spent by experts while

**Listing 1** Example Moirai configuration

```
[Cluster]
machines = attacker, victim

[attacker]
box = TFDuesing/Fedora-20
ip = 192.168.51.5

[victim]
box = TFDuesing/Fedora-20
ip = 192.168.51.100

[Scenario]
tasks = botmaster, shellshock, pupy
duration = 10m

[botmaster]
target = attacker
timing = 0
actions = ./botmaster.py
files = botmaster.py

[shellshock]
target = attacker
timing = 1m
actions = ./shellshock.sh 192.168.51.100
files = shellshock.sh

[pupy]
target = attacker
timing = +5m
actions = ./download-pupy.sh
files = download-pupy.sh
```

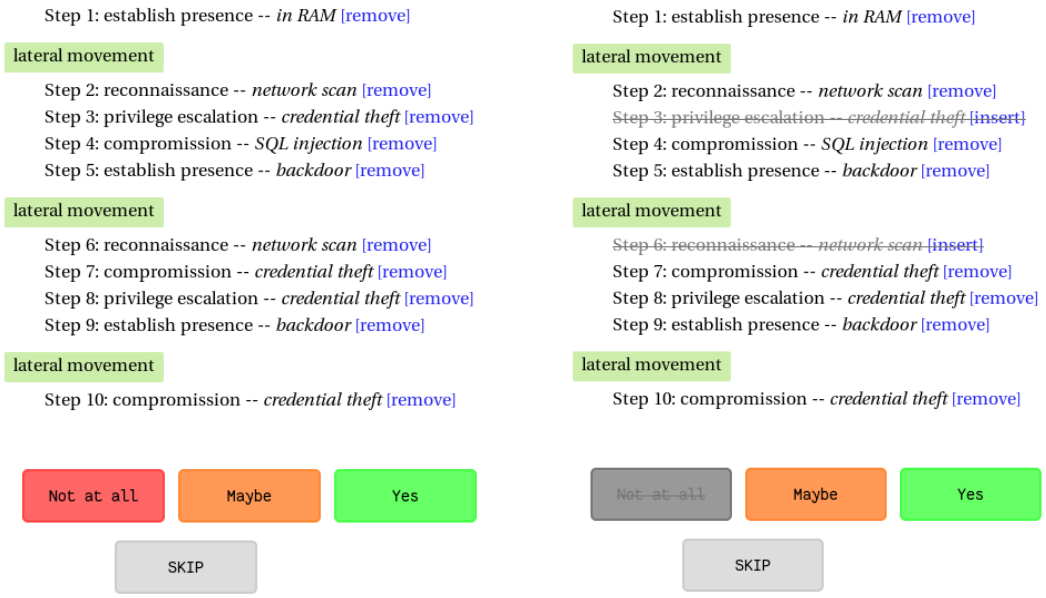
increasing the number of APT like attack scenarios at our disposal.

In order to increase experts' engagement and in the hope of obtaining more rated scenarios, we added gamification elements. Experts could chose a pseudonym and a score, displayed on a leaderboard, was associated with their name. The score is an agreement score between raters [Banerjee et al. 1999], and is useful in our own rating of the raters. In particular, if we see that some raters agree with each other on some models, we can have more confidence in the rating. On the other hand, if we see that a rater never agrees with other raters we can decide to ignore their results. In order to be able to compute a useful score, we had to make two assumptions. First, while the models are generated at random, each rater had a chance to rate an already rated scenario at each step instead of getting a newly generated one. This is necessary because otherwise, different raters would almost never rate the same scenarios and we could not compute the score. The second assumption we made is that if one rater removed some steps of a scenario and marked it as an APT, then, the next rater who marked it as an APT would remove the same steps. In other words, when computing the score, we ignored whether steps had been removed or not. This is, once again, necessary to be able to compute the score. Otherwise, two raters removing different steps of the same attack scenarios could not be compared. This score is not only used for the gamification but is also crucial to our evaluation of the results and of the raters. Finally, once we have collected enough data for the creation of the state transition matrix, we can also use the best rated scenarios as inspiration for the creation of evaluation scenarios to play with Moirai.

## 5.4 Data generation

### 5.4.1 Creating a scenario

With all the tools in place, we must now setup an APT scenario that we can use to evaluate the solutions we propose in this thesis. The first when creating a scenario is choosing a sequence of attacks for the APT we want to represent. In this scenario we keep it simple. We use the typical APT life cycle presented in Section 1.2 on page 3, and remove the cycles. We are left with the following steps: "reconnaissance", "compromission", "establish presence", "privilege escalation" and "mission completion". We can then decide what to use at each step. In our case, since the scenario was created a short while after the shellshock attack became famous, this is the vector we decided to use for the "compromission". This means that the target has to be an apache web server. For the "establish presence", we use Remote Access Tool (RAT) which connects




(a) A scenario with all the steps selected. The raters can choose any of the colored answer or even skip the scenario.

(b) A scenario with some steps removed. Notice the stroked out steps in gray and the grayed out "Not at all" button.

Figure 5.2: Screenshots of the web application

to IRC and waits for instructions there. This tool was found by one of our business partners in the wild, so it is a real world RAT. For the "privilege escalation", we can code a small program which uses to fact that the apache server is slightly mis-configured to open a root level shell. That shell can then be used to export the database as well as the file containing the system's users' passwords. We now have the working core of the scenario. We start by implementing this core. We create one victim Virtual Machine (VM) and one VM for the attacker. The victim's VM must have a version of apache that is vulnerable to the shellshock attack. The attacker's VM must run an IRC server so that the RAT can connect to it. We create provisioning scripts for each VM so that their configuration can be automated. Once the VM are setup, we must automate each step of the attack chain by writing scripts. After the scripts are written, we can check that the scenario works by running it by hand, i.e. by triggering each step of each attack by hand. If some of the scripts require additional packages or configurations, we must add that to the provisioning scripts. Once we have verified that the APT campaign works when each phase is triggered independently, we can automate it with Moirai. We want to make sure that we can play the whole scenario with a single command.

Once we have the base scenario working and automated, we want to add noise. The idea is that in a normal system, we would expect every day users doing completely normal

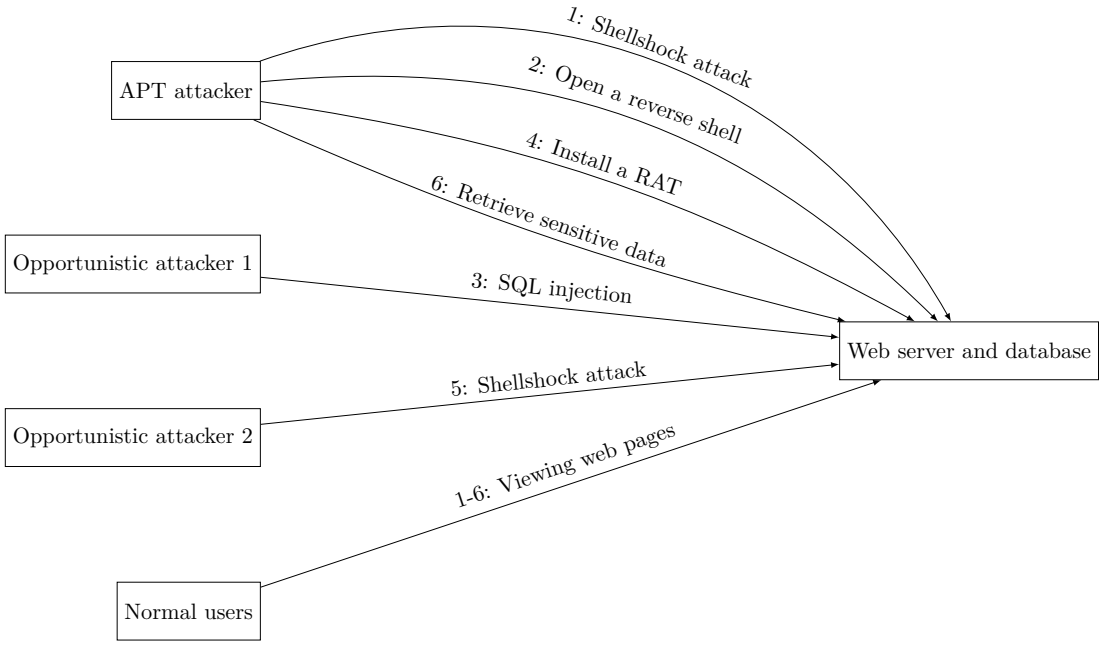



Figure 5.3: The complete scenario base, with one APT, several other attackers and normal traffic.

things as well as other attackers. Even if the attackers are not full-blown APT campaigns, we cannot assume that the system will only ever be under attack by one attacker at a time. To simulate normal website users, we can use web crawlers and website benchmark utilities to program semi-random page views. These page views simulate several users so they should be able to happen at the same time. For the additional attackers, we will create two, each with their own attack. The first one will use an SQL injection to dump the database of the website; we can add that vulnerability to the website. The second one will use the shellshock vulnerability, it is a well known and powerful vulnerability, to download the system's users' password file. We script those attacks and add the required configuration to the provisioning scripts. Once both attacks and the normal traffic simulation work, we add them to the Moirai scenario. We end up with the scenario described in Figure 5.3.

### 5.4.2 Generating raw data

Since the scenario is now fully defined, including provisioning scripts for the VMs and Moirai configuration, we only need to add the IDS to test. We install it on the web server being attacked and make sure it starts when the VM does. Note that we could instead have Moirai start it at the beginning of the scenario, but this means modifying

the Moirai configuration file for each IDS to test which we try to avoid. Once the VM are ready, we can play the scenario with a single `moirai spin` command. We can then retrieve the IDS' logs after the scenario finishes. This could also be done automatically by Moirai but it would entail changing the configuration for each IDS.

### 5.4.3 Preparing the raw data for exploitation

Ideally, the data outputted by the IDS could be used without changes as the input to our IFT and HMM modules. However the part of the IDS taking the data from the probes and finding the individual attacks has not been completely implemented. This means that we have to take the raw data from the probes and emulate the part that finds the individual attacks. This can be done because we know exactly which attacks are played in the scenario. For each attack, we can define the earliest signs which should be detected as the individual attack. Then, in the raw data, we can find the signs of every attack and mark it as if the IDS had indeed detected them.

# Chapter 6

# Concluding discussion

In the introduction, we presented Advanced Persistent Threats. While attack campaigns are not new, the fact that these attack campaigns are performed by highly skilled actors with powerful backings certainly makes them a bigger threat. Current Intrusion Detection Systems are beginning to try to detect them but there are a lot of challenges remaining. Akheros is a company created specifically to design an IDS capable of detecting them. The project is divided in three thesis. The first one, by Matthieu Hourbracq, concentrates on modeling the system as it evolves in an unsupervised manner. This model of the behaviours of the system is then used to detect incongruous events, i.e. events that cannot be explained by any of the known behaviours of the system and thus could not have been anticipated. The results are interesting and form the base of the Akheros IDS. The thesis will be defended soon. The second thesis analyses these incongruous events to check if they are a security risk. Events that are deemed a security risk are seen as attacks and are classified. This is being developed by Mark Angoustures and shows promise too. It will also be defended soon. The third thesis, this thesis, looks for links between attacks in order to reconstruct attack campaigns. This is done in a two-step process and results in a list of potential attack campaigns ranked from most probably an APT to least probably an APT.

In the first step of our approach, we use Information Flow Tracking to link related attacks together. This gives us potential attack campaigns. In contrast with the IFT schemes presented in Section 2.2 on page 22 where it is used either to detect policy violations or to analyse a single malware, our contribution is showing that IFT can be used to link attacks that are part of the same attack campaign. As far as we can tell, this is the first time that IFT is used as a forensic tool in order to highlight whole attack campaigns, such as APTs. Compared to traditional forensic approaches where a human operator looks for clues once a severe attack has had an impact on the system, using IFT means that the forensic is being done in real time. This is especially important since attackers remove as many signs of their presence as possible. A human operator

examining a system after an attack will only be able to find clues that the attackers forgot to remove. Our method instead analyses the system in real time so that it does not matter if attackers later clean up their traces. In addition, manual forensic takes time which means that an analyst will not be deployed for every small attack. Our always-on method will be able to link related attacks whatever their impact. By linking several seemingly low-impact attacks as part of the same campaign, their actual impact can be more properly assessed, and a human operator deployed before an attack with a large impact on the system is executed as part of this campaign. On top of all this, our IFT-based method is not dependant on the time elapsed between attacks: it is able to link related attacks whether five minutes or five months elapsed between the two attacks. By comparison, a human operator would have a lot more things to check if five months had elapsed and finding the link would take much longer and would be less likely produce results. In this way, IFT acts like a timeless memory, able to follow the flows of information months or years after they start and for a small cost: storing the taints. Finally, as part of the Akheros IDS, tracking flows of information at the OS level has the advantage that our probes already collect all the necessary information, which means that there is no tracking overhead on top of the IDS and that the only overhead is the propagation of the taints which is not computationally intensive. In terms of deployment, the probes are in kernel module which can be compiled separately and activated at runtime. However, this approach does have false positives. Collecting data at the OS level means we do not follow flows of information inside a process, whereas CPU, compiler and code based approaches are capable of doing so. This is a trade-off we accepted because of the ease of deployment which is primordial for the Akheros use case. False positives manifests as unrelated attacks being linked together, and in order to reduce their numbers, we added a second step to our approach.

The second step is there to filter the real attack campaign from the erroneously linked attacks. This is done using a Hidden Markov Model. Our contribution regarding HMM is twofold. First we show that APTs can be modeled with an HMM. This was already done by Ourston et al. in [Ourston et al. 2003] for multi-step attacks, and we simply show that even though APT are executed by skilled attackers, they can still be modeled by an HMM. The second, and more novel, contribution is the definition of a new score to rank potential attack campaigns from most likely an APT to least likely an APT. This score has two properties that other scores do not possess and which necessitated its creation. The first is that it is used to compare how different chains fit the model. Usually, scores, even recent ones such as the HBIC [Biem, Ha, and Subrahmonia 2002] and the DIC [Biem 2003] are used to compare how different models are fitted by a single chain. The main difference between the two use cases is that in our case, we must be able to compare

chains of different lengths. Our score is the only score, as far as we are aware, that can be used to compare the fit of chains of different lengths. The second property is that our score takes into account potentially missing observations. Since APTs are executed by skilled actors, even advanced IDS will probably not detect every attack. We created our score with this in mind, and as far as we know, this is the only score where we can specify a probability of missing an observation and the score takes that into account. In order to establish the matrices of the HMM itself, we use data from two main sources. The observation matrix is created from the APT reports that are freely available on the web. The transition matrix and the initial probabilities are obtained by polling experts. This score, with the first step of linking attacks through IFT, shores up a weakness of the approach of Ourston et al. in [Ourston et al. 2003] which is also present in the improved version of Q. Zhang, Man, and Wu in [Q. Zhang, Man, and Wu 2009]: these approaches only work if all the attacks have been detected and are part of the same attack campaign. In contrast, our approach only analyses chains of attacks found by IFT and takes into account the probability of missing observations.

There is a third contribution in this thesis: Moirai. Moirai is a tool we designed to orchestrate scenarios in order to test our approach for identifying APTs. The motivation behind this tool is the realisation that, while there are reference datasets for evaluating IDS, they are often criticised, and none of them were made with APTs in mind. This is well summarised by Małowidzki, Bereziński, and Mazur in [Małowidzki, Bereziński, and Mazur 2015]. In addition, using a dataset forces the IDS to use the same sources that are included in the dataset. In our case, if the dataset does not include the information necessary for IFT, then we cannot use it. While there are tools for generating evaluation data, they are specific. For example, the tool proposed by Béla Genge et al. in [Béla Genge et al. 2012] targets Industrial Control Systems while the tool proposed by Sommers, Yegneswaran, and Barford in [Sommers, Yegneswaran, and Barford 2004] is specific to network based IDS. With all this in mind, we decided to instead design a generic tool for playing scenarios. With the advances in network and virtualisation, it is now easy to share Virtual Machines (VMs). As a matter of fact, Vagrant is a tool created for this exact purpose: it can download VMs from a remote repository, set it up and launch it. We based Moirai on top of this tool. In addition to Vagrant capabilities for setting up and configuring VMs, Moirai is capable of executing arbitrary actions at set times on those VMs. The scenarios used by Moirai are described in a single text file and can thus easily be shared. By using VMs, the scenario will be executed in the same conditions each time. The only difference, from one execution to the next, is the IDS being evaluated. This means that we can now compare IDS even if they use different sources of data as the IDS will be run in the evaluating environment and will be free to

collect its own data however it requires.

Scenarios played with Moirai are used to evaluate the two steps presented above. The scenarios are played with our probes installed on the system to protect. In our evaluation, we have shown that IFT is capable of finding the links between attacks part of the same campaign. However, we also see that IFT also finds links between unrelated attacks. This motivates the second step of the process: using an HMM to find which potential attack campaigns highlighted through IFT are more probably APTs. In evaluation, we show that the base score we created is capable of ordering APTs from most probably to least probably an APT regardless of the length of the chain; this was the first reason for designing our own score. The second reason was to take into account possible missing observations, since we cannot guarantee that every attack will be detected. IFT is relatively immune to undetected attacks since flows of information are always measured, but HMM is sensitive to missing observations. The evaluation shows that our full score can take into account possible missing observations given a probability of missing an observation. The point is to give a better score to an attack campaign if it is potentially only missing a few observations.

### Future work

The results obtained by our various contributions are interesting. However, they are still several areas that could be furthered.

The first point, and the weak point of this thesis, is the evaluation. While the approaches have been evaluated on a complex scenario, the number of different scenarios is lacking. In particular there are no large scenarios. We want to create a scenario containing at least two APTs happening at the same time and frequent unrelated attacks. Moirai makes sharing and replaying scenarios easy, but the creation of each scenario still requires time. In addition, in order to be able to use existing malware in scenarios, Moirai must be further developed to include network spoofing for example. Some existing RAT software will always try to connect to the same C&C server with no way to configure it; thus, Moirai must enable us to easily and transparently redirect the network traffic intended for that C&C server to a VM part of the scenario. By sharing Moirai with the community, we are hoping that more scenarios are created and enhanced which would benefit the whole community. While the initial response to the Moirai presentation was good, we are not aware, at this time, of other users.

The second area that we would like to improve is the integration between the two steps. For now, the two steps work mostly independently. In particular, the IFT step does not know when the HMM step finds that a potential attack campaign is not, actually, an

attack campaign. Since the HMM step is designed to filter the real attack campaigns from the false positives, it would be interesting to add a feedback mechanism so that the output of the HMM would be used by the IFT step to remove taints and sever erroneous links between attacks. This should reduce the amount of false positives in the IFT step which should make the filtering through the HMM easier.

The third idea that we want to explore is the addition of a feedback loop to the HMM. Our current implementation does not include learning of the model while the HMM is active. We would like to add a feedback mechanism so that a human operator could indicate to the HMM which chains of attacks are really APTs. This feedback would be incorporated into the model to improve it. The advantages of this would be that the matrices established from the experts and the APT reports would only be the seed of the model. Once installed in a specific system, the model would evolve with each APT targeting the system and would be better tailored to protecting that specific system, while also following the evolution of the APT as time passes.

# Bibliography

[Ács-Kurucz et al. 2014] Ács-Kurucz, Gábor, Balázs, Zoltán, Bencsáth, Boldizsár, Buttyán, Levente, Kamarás, Roland, Molnár, Gábor, and Vaspöri, Gábor; *An independent test of APT attack detection appliances*; 2014; URL: `https://blog.mrg-effitas.com/wp-content/uploads/2014/11/Crysys_MRG_APT_detection_test_2014.pdf` (cit. on p. 15).

[Aguirre-Hernández and Farewell 2002] Aguirre-Hernández, Rebeca and Farewell, VT; "A Pearson-type goodness-of-fit test for stationary and time-continuous Markov regression models"; Statistics in medicine 21.13 (2002), 1899–1911 (cit. on p. 30).

[Akaike 1974] Akaike, Hirotugu; "A new look at the statistical model identification"; IEEE transactions on automatic control 19.6 (1974), 716–723 (cit. on pp. 28, 59).

[Akheros 2016] Akheros; *Moirai*; 2016; URL: `https://github.com/akheros/moirai` (cit. on pp. xv, xvii, 36, 82).

[Akheros 2016] – ; *Scenarios for Moirai*; 2016; URL: `https://github.com/akheros/moirai-scenarios` (cit. on pp. xvi, xvii, 37, 82).

[Andriatsimandefitra and V. V. T. Tong 2014] Andriatsimandefitra, Radoniaina and Tong, Valérie Viet Triem; "Capturing Android malware behaviour using system flow graph"; *Network and System Security*; Springer, 2014, 534–541 (cit. on p. 26).

[Angoustures et al. 2017] Angoustures, Mark, Erra, Robert, and Di Bernardino, Elena; *A quick malware detection algorithm using a suspiciousness score of binaries*; Preprint.; 2017 (cit. on p. 6).

[Ansible 2012] Ansible; *Ansible*; 2012; URL: `https://www.ansible.com/` (cit. on p. 82).

[Arnes et al. 2006] Arnes, André, Valeur, Fredrik, Vigna, Giovanni, and Kemmerer, Richard A; "Using hidden Markov models to evaluate the risks of intrusions: system architecture and model validation"; *Recent Advances in Intrusion Detection*; Springer; 2006, 145–164 (cit. on pp. xii, 32).

[Banerjee et al. 1999] Banerjee, Mousumi, Capozzoli, Michelle, McSweeney, Laura, and Sinha, Debajyoti; "Beyond kappa: A review of interrater agreement measures"; Canadian journal of statistics 27.1 (1999), 3–23 (cit. on p. 85).

[Baum and Petrie 1966] Baum, Leonard E and Petrie, Ted; "Statistical inference for probabilistic functions of finite state Markov chains"; The annals of mathematical statistics 37.6 (1966), 1554–1563 (cit. on p. 28).

[Baumard 2015] Baumard, Philippe; *Autonomous detection of incongruous behaviors*; US Patent App. 14/660,519; 2015 (cit. on p. 5).

[Bhatt et al. 2014] Bhatt, Parth, Toshiro Yano, Edgar, and Gustavsson, Per M; "Towards a Framework to Detect Multi-stage Advanced Persistent Threats Attacks"; *Service Oriented System Engineering (SOSE), 2014 IEEE 8th International Symposium on*; IEEE; 2014, 390–395 (cit. on p. 18).

[Biem 2003] Biem, Alain; "A model selection criterion for classification: Application to HMM topology optimization"; *Document Analysis and Recognition, 2003. Proceedings. Seventh International Conference on*; IEEE; 2003, 104–108 (cit. on pp. 29, 90).

[Biem et al. 2002] Biem, Alain, Ha, Jin-Young, and Subrahmonia, Jayashre; "A Bayesian model selection criterion for HMM topology optimization"; *Acoustics, Speech, and Signal Processing (ICASSP), 2002 IEEE International Conference on*; vol. 1; IEEE; 2002, I–989 (cit. on pp. 29, 90).

[Binde et al. 2011] Binde, Beth, McRee, Russ, and O'Connor, Terrence J; "Assessing outbound traffic to uncover advanced persistent threat"; *SANS Institute. Whitepaper*; 2011 (cit. on p. 16).

[Bowers et al. 2012] Bowers, Kevin D, Van Dijk, Marten, Griffin, Robert, Juels, Ari, Oprea, Alina, Rivest, Ronald L, and Triandopoulos, Nikos; "Defending against the unknown enemy: Applying FlipIt to system security"; *Decision and Game Theory for Security*; Springer, 2012, 248–263 (cit. on p. 13).

[Brogi 2016] Brogi, Guillaume; *APT models*; 2016; URL: `https://aptmodels-guiniol.rhcloud.com/` (cit. on p. 83).

[Brogi and Di Bernardino 2018] Brogi, Guillaume and Di Bernardino, Elena; *Hidden Markov models for advanced persistent threats*; Submitted to the International Journal of Security and Networks by Inderscience, under review.; 2018 (cit. on pp. xvi, 8).

[Brogi and Viet Triem Tong 2017] Brogi, Guillaume and Viet Triem Tong, Valérie; "Sharing and replaying attack scenarios with Moirai"; *RESSI 2017: Rendez-vous de la Recherche et de l'Enseignement de la Sécurité des Systèmes d'Information*; 2017 (cit. on pp. xvii, 8).

[Brogi and Viet Triem Tong 2016] – ; "TerminAPTor: Highlighting Advanced Persistent Threats through Information Flow Tracking"; *New Technologies, Mobility and Security (NTMS), 2016 8th IFIP International Conference on*; IEEE; 2016, 1–5 (cit. on pp. xvi, 8).

[Bukac et al. 2014] Bukac, Vit, Lorenc, Vaclav, and Matyáš, Vashek; "Red queen's race: APT win-win game"; *Cambridge International Workshop on Security Protocols*; Springer; 2014, 55–61 (cit. on p. 19).

[Chandran et al. 2015] Chandran, Saranya, Hrudya, P, and Poornachandran, Prabaharan; "An efficient classification model for detecting advanced persistent threat"; *Advances in Computing, Communications and Informatics (ICACCI), 2015 International Conference on*; IEEE; 2015, 2001–2009 (cit. on pp. viii, 16).

[C.-M. Chen et al. 2012] Chen, Chia-Mei, Guan, DJ, Huang, Yu-Zhi, and Ou, Ya-Hui; "Attack Sequence Detection in Cloud Using Hidden Markov Model"; *Information Security (Asia JCIS), 2012 Seventh Asia Joint Conference on*; IEEE; 2012, 100–103 (cit. on p. 33).

[C.-M. Chen et al. 2013] Chen, Chia-Mei, Hsiao, Han-Wei, Yang, Peng-Yu, and Ou, Ya-Hui; "Defending malicious attacks in cyber physical systems"; *Cyber-Physical Systems, Networks, and Applications (CPSNA), 2013 IEEE 1st International Conference on*; IEEE; 2013, 13–18 (cit. on p. 33).

[C.-M. Chen et al. 2014] Chen, Chia-Mei, Yang, Peng-Yu, Ou, Ya-Hui, and Hsiao, Han-Wei; "Targeted Attack Prevention at Early Stage"; *Advanced Information Networking and Applications Workshops (WAINA), 2014 28th International Conference on*; IEEE; 2014, 866–870 (cit. on p. 33).

[Z. Chen et al. 2016] Chen, Zhaomin, Yeo, Chai Kiat, Francis, Bu Sung Lee, and Lau, Chiew Tong; "Combining MIC feature selection and feature-based MSPCA for network traffic anomaly detection"; *Digital Information Processing, Data Mining, and Wireless Communications (DIPDMWC), 2016 Third International Conference on*; IEEE; 2016, 176–181 (cit. on p. 35).

[Cordero et al. 2015] Cordero, Carlos Garcia, Vasilomanolakis, Emmanouil, Milanov, Nikolay, Koch, Christian, Hausheer, David, and Mühlhäuser, Max; "ID2T: A DIY dataset creation toolkit for Intrusion Detection Systems"; *Communications and Network Security (CNS), 2015 IEEE Conference on*; IEEE; 2015, 739–740 (cit. on p. 35).

[Dalton et al. 2007] Dalton, Michael, Kannan, Hari, and Kozyrakis, Christos; "Raksha: a flexible information flow architecture for software security"; *ACM SIGARCH Computer Architecture News*; vol. 35; 2; ACM; 2007, 482–493 (cit. on pp. 23, 26).

[De Vries et al. 2012] De Vries, Jelle, Hoogstraaten, Hans, Berg, Jan van den, and Daskapan, Semir; "Systems for Detecting Advanced Persistent Threats: A Development Roadmap Using Intelligent Data Analysis"; *Cyber Security (CyberSecurity), 2012 International Conference on*; IEEE; 2012, 54–61 (cit. on p. 19).

[DARPA 2016] Defense Advanced Research Projects Agency; *Impact*; 2016; URL: `https://www.impactcybertrust.org/` (cit. on p. 35).

[Devijver 1985] Devijver, Pierre A; "Baum's forward-backward algorithm revisited"; Pattern Recognition Letters 3.6 (1985), 369–373 (cit. on pp. 27, 57).

[Enck et al. 2014] Enck, William, Gilbert, Peter, Han, Seungyeop, Tendulkar, Vasant, Chun, Byung-Gon, Cox, Landon P, Jung, Jaeyeon, McDaniel, Patrick, and Sheth, Anmol N; "TaintDroid: an information-flow tracking system for realtime privacy monitoring on smartphones"; ACM Transactions on Computer Systems (TOCS) 32.2 (2014), 5 (cit. on pp. x, 25).

[Engen 2010] Engen, Vegard; "Machine learning for network based intrusion detection"; PhD thesis; Bournemouth University, 2010 (cit. on p. 35).

[Fireeye Labs 2015] Fireeye Labs; *APT30 and the Mechanics of a Long-Running Cyber Espionage Operation*; 2015; URL: `https://www2.fireeye.com/rs/fireye/images/rpt-apt30.pdf` (cit. on pp. 12, 13, 83).

[Flores et al. 2009] Flores, Juan J, Antolino, Anastacio, and Garcia, Juan M; "Evolving Hidden Markov Models For Network Anomaly Detection"; Book: Artificial Intelligence & Applications: Hybrid Intelligent Systems, Intelligent Learning Environments, and Computer Security. Editor: Alexander Gelbukh. SMIA. ISBN (2009), 978–607 (cit. on p. 30).

[Flores et al. 2015] Flores, Juan J, Calderon, Felix, Antolino, Anastacio, and Garcia, Juan M; "Network Anomaly Detection by Continuous Hidden Markov Models: An Evolutionary Programming Approach"; Intelligent Data Analysis 1 (2015), 1 (cit. on p. 30).

[Forcepoint 2016] Forcepoint; *Monsoon - Analysis of an APT Campaign*; 2016; URL: `https://www.forcepoint.com/sites/default/files/resources/files/forcepoint-security-labs-monsoon-analysis-report.pdf` (cit. on pp. viii, 4).

[Forney 1973] Forney, G David; "The Viterbi algorithm"; Proceedings of the IEEE 61.3 (1973), 268–278 (cit. on pp. 27, 57, 60).

[Friedberg et al. 2015] Friedberg, Ivo, Skopik, Florian, Settanni, Giuseppe, and Fiedler, Roman; "Combating advanced persistent threats: from network event correlation to incident detection"; Computers & Security 48 (2015), 35–57 (cit. on p. 17).

[Gao et al. 2003] Gao, Fei, Sun, Jizhou, and Wei, Zunce; "The prediction role of hidden Markov model in intrusion detection"; *Electrical and Computer Engineering, 2003. IEEE CCECE 2003. Canadian Conference on*; vol. 2; IEEE; 2003, 893–896 (cit. on p. 34).

[Béla Genge et al. 2012] Genge, Béla, Siaterlis, Christos, Fovino, Igor Nai, and Masera, Marcelo; "A cyber-physical experimentation environment for the security analysis of networked industrial control systems"; Computers & Electrical Engineering 38.5 (2012), 1146–1161 (cit. on pp. 35, 91).

[Giura and W. Wang 2012] Giura, Paul and Wang, Wei; "A context-based detection framework for advanced persistent threats"; *Cyber Security (CyberSecurity), 2012 International Conference on*; IEEE; 2012, 69–74 (cit. on pp. viii, 19, 21).

[Gladyshev and Patel 2004] Gladyshev, Pavel and Patel, Ahmed; "Finite state machine approach to digital event reconstruction"; Digital Investigation 1.2 (2004), 130–149 (cit. on p. 21).

[Godefroy et al. 2014] Godefroy, Erwan, Totel, Eric, Majorczyk, Frédéric, and Hurfin, Michel; "Génération automatique de règles de corrélation pour la détection d'attaques complexes"; *9eme conférence sur la Sécurité des Architectures Réseaux et des Systèmes d'Information (SAR-SSI)*; 2014, 10 (cit. on p. 15).

[Haines et al. 2001] Haines, Joshua W, Lippmann, Richard P, Fried, David J, Zissman, MA, and Tran, E; *1999 DARPA intrusion detection evaluation: Design and procedures*; tech. rep.; DTIC Document, 2001 (cit. on p. 34).

[HashiCorp 2010] HashiCorp; *Vagrant*; 2010; URL: `https://www.vagrantup.com/` (cit. on pp. xv, 82).

[Haslum et al. 2007] Haslum, Kjetil, Abraham, Ajith, and Knapskog, Svein; "Dips: A framework for distributed intrusion prediction and prevention using hidden Markov models and online fuzzy risk assessment"; *Information Assurance and Security, 2007. IAS 2007. Third International Symposium on*; IEEE; 2007, 183–190 (cit. on p. 32).

[Haslum et al. 2008] – ; "Fuzzy online risk assessment for distributed intrusion prediction and prevention systems"; *Computer Modeling and Simulation, 2008. UKSIM 2008. Tenth International Conference on*; IEEE; 2008, 216–223 (cit. on p. 33).

[Haslum et al. 2008] Haslum, Kjetil, Moe, Marie Elisabeth Gaup, and Knapskog, Svein J; "Real-time intrusion prevention and security analysis of networks using HMMs"; *Local Computer Networks, 2008. LCN 2008. 33rd IEEE Conference on*; IEEE; 2008, 927–934 (cit. on p. 33).

[Hauser et al. 2013] Hauser, Christophe, Tronel, Frédéric, Fidge, Colin, and Mé, Ludovic; "Intrusion detection in distributed systems, an approach based on taint marking"; *2013 IEEE International Conference on Communications (ICC)*; IEEE; 2013, 1962–1967 (cit. on p. 25).

[Hauser et al. 2012] Hauser, Christophe, Tronel, Frédéric, Reid, Jason, and Fidge, Colin; "A taint marking approach to confidentiality violation detection"; *Proceedings of the Tenth Australasian Information Security Conference-Volume 125*; Australian Computer Society, Inc.; 2012, 83–90 (cit. on pp. x, 25).

[Hernando et al. 2005] Hernando, Diego, Crespi, Valentino, and Cybenko, George; "Efficient computation of the hidden Markov model entropy for a given observation sequence"; Information Theory, IEEE Transactions on 51.7 (2005), 2681–2685 (cit. on p. 28).

[Hourbracq et al. 2017] Hourbracq, Matthieu, Wuillemin, Pierre-Henri, Gonzales, Christophe, and Baumard, Philippe; "Learning and selection of dynamic Bayesian Networks for non-stationary processes in real time"; *30th International Florida AI Research Society Conference, FLAIRS-30*; Marco Island, United States, May 2017 (cit. on p. 6).

[Hourbracq et al. 2016] – ; "Real Time Learning of Non-stationary Processes with Dynamic Bayesian Networks"; *International Conference on Information Processing and Management of Uncertainty in Knowledge-Based Systems*; Springer; 2016, 338–350 (cit. on p. 6).

[Hutchins et al. 2011] Hutchins, Eric M, Cloppert, Michael J, and Amin, Rohan M; "Intelligence-driven computer network defense informed by analysis of adversary campaigns and intrusion kill chains"; Leading Issues in Information Warfare & Security Research 1 (2011), 80 (cit. on p. 18).

[Ioannou et al. 2013] Ioannou, Georgios, Louvieris, Panos, Clewley, Natalie, and Powell, Gavin; "A Markov multi-phase transferable belief model: an application for predicting data exfiltration APTs"; *Information Fusion (FUSION), 2013 16th International Conference on*; IEEE; 2013, 842–849 (cit. on p. 31).

[Jain and Abouzakhar 2012] Jain, Ruchi and Abouzakhar, Nasser S; "Hidden Markov model based anomaly intrusion detection"; *Internet Technology And Secured Transactions, 2012 International Conference for*; IEEE; 2012, 528–533 (cit. on p. 31).

[Katipally et al. 2011] Katipally, Rajeshwar, Yang, Li, and Liu, Anyi; "Attacker behavior analysis in multi-stage attack detection system"; *Proceedings of the Seventh Annual Workshop on Cyber Security and Information Intelligence Research*; ACM; 2011, 63 (cit. on p. 34).

[Keniston et al. 2005] Keniston, Jim, Panchamukhi, Prasanna S, and Hiramatsu, Masami; *Kernel Probes (Kprobes)*; 2005; URL: `https://www.kernel.org/doc/Documentation/kprobes.txt` (cit. on p. 80).

[Khanna and H. Liu 2008] Khanna, Rahul and Liu, Huaping; "Control theoretic approach to intrusion detection using a distributed hidden Markov model"; Wireless Communications, IEEE 15.4 (2008), 24–33 (cit. on p. 32).

[Khanna and H. Liu 2006] – ; "System approach to intrusion detection using hidden Markov model"; *Proceedings of the 2006 international conference on Wireless communications and mobile computing*; ACM; 2006, 349–354 (cit. on p. 32).

[Koch et al. 2014] Koch, Robert, Golling, Mario, and Rodosek, Gabi Dreo; "Towards Comparability of Intrusion Detection Systems: New Data Sets"; *TERENA Networking Conference*; 2014, 7 (cit. on p. 35).

[L. C. Lam and Chiueh 2006] Lam, Lap Chung and Chiueh, Tzi-cker; "A general dynamic information flow tracking framework for security applications"; *Computer Security*

*Applications Conference, 2006. ACSAC'06. 22nd Annual*; IEEE; 2006, 463–472 (cit. on p. 23).

[M. S. Lam et al. 2008] Lam, Monica S, Martin, Michael, Livshits, Benjamin, and Whaley, John; "Securing web applications with static and dynamic information flow tracking"; *Proceedings of the 2008 ACM SIGPLAN symposium on Partial evaluation and semantics-based program manipulation*; ACM; 2008, 3–12 (cit. on p. 25).

[Li et al. 2011] Li, Frankie, Lai, Anthony, and DDL; "Evidence of Advanced Persistent Threat: A case study of malware for political espionage"; *Malicious and Unwanted Software (MALWARE), 2011 6th International Conference on*; IEEE; 2011, 102–109 (cit. on p. 15).

[Lippmann et al. 2000] Lippmann, Richard, Haines, Joshua W, Fried, David J, Korba, Jonathan, and Das, Kumar; "The 1999 DARPA off-line intrusion detection evaluation"; Computer networks 34.4 (2000), 579–595 (cit. on p. 34).

[Little et al. 2016] Little, Anna, Mountrouidou, Xenia, and Moseley, Daniel; "Spectral Clustering Technique for Classifying Network Attacks"; *Big Data Security on Cloud (BigDataSecurity), IEEE International Conference on High Performance and Smart Computing (HPSC), and IEEE International Conference on Intelligent Data and Security (IDS), 2016 IEEE 2nd International Conference on*; IEEE; 2016, 406–411 (cit. on p. 35).

[C. Liu et al. 2012] Liu, Changwei, Singhal, Anoop, and Wijesekera, Duminda; "Using attack graphs in forensic examinations"; *Availability, Reliability and Security (ARES), 2012 Seventh International Conference on*; IEEE; 2012, 596–603 (cit. on p. 21).

[MacKay Altman 2004] MacKay Altman, Rachel; "Assessing the Goodness-of-Fit of Hidden Markov Models"; Biometrics 60.2 (2004), 444–450 (cit. on p. 30).

[Maggi et al. 2008] Maggi, Federico, Zanero, Stefano, and Iozzo, Vincenzo; "Seeing the invisible: forensic uses of anomaly detection and machine learning"; ACM SIGOPS Operating systems review 42.3 (2008), 51–58 (cit. on p. 33).

[Mahoney and Chan 2003] Mahoney, Matthew V and Chan, Philip K; "An analysis of the 1999 DARPA/Lincoln Laboratory evaluation data for network anomaly detection"; *International Workshop on Recent Advances in Intrusion Detection*; Springer; 2003, 220–237 (cit. on p. 35).

[Małowidzki et al. 2015] Małowidzki, Marek, Bereziński, Przemysław, and Mazur, Michał; "Network Intrusion Detection: Half a Kingdom for a Good Dataset"; *Proceedings of NATO STO SAS-139 Workshop, Portugal*; 2015 (cit. on pp. xiv, 35, 91).

[Mandiant Intelligence Center 2013] Mandiant Intelligence Center; *APT1: Exposing one of China's cyber espionage units*; 2013; URL: `http://intelreport.mandiant.com/Mandiant_APT1_Report.pdf` (cit. on pp. 12, 13, 83).

[Maxion and Tan 2000] Maxion, Roy A and Tan, Kymie MC; "Benchmarking anomaly-based detection systems"; *Dependable Systems and Networks, 2000. DSN 2000. Proceedings International Conference on*; IEEE; 2000, 623–630 (cit. on p. 35).

[McAfee Labs 2013] McAfee Labs; *Operation Red October*; 2013; URL: `https://kc.mcafee.com/resources/sites/MCAFEE/content/live/PRODUCT_DOCUMENTATION/24000/PD24250/en_US/McAfee_Labs_Threat_Advisory_Exploit_Operation_Red_Oct.pdf` (cit. on pp. 12, 13, 83).

[McHugh 2000] McHugh, John; "The 1998 Lincoln laboratory ids evaluation"; *International Workshop on Recent Advances in Intrusion Detection*; Springer; 2000, 145–161 (cit. on p. 35).

[Mirkovic et al. 2006] Mirkovic, Jelena, Arikan, Erinc, Wei, Songjie, Fahmy, Sonia, Thomas, Roshan, and Reiher, Peter; "Benchmarks for DDoS defense evaluation"; *Military Communications Conference, 2006. MILCOM 2006. IEEE*; IEEE; 2006, 1–10 (cit. on p. 36).

[Ourston et al. 2003] Ourston, Dirk, Matzner, Sara, Stump, William, and Hopkins, Bryan; "Applications of hidden Markov models to detecting multi-stage network attacks"; *System Sciences, 2003. Proceedings of the 36th Annual Hawaii International Conference on*; IEEE; 2003, 10–pp (cit. on pp. xii, 32, 90, 91).

[Posada and Buckley 2004] Posada, David and Buckley, Thomas R; "Model selection and model averaging in phylogenetics: advantages of Akaike information criterion and Bayesian approaches over likelihood ratio tests"; Systematic biology 53.5 (2004), 793–808 (cit. on p. 29).

[Qian et al. 2006] Qian, Jun, Xu, Chao, and Shi, Meilin; "Redesign and implementation of evaluation dataset for intrusion detection system"; *Emerging Trends in Information and Communication Security*; Springer, 2006, 451–465 (cit. on p. 35).

[Qin et al. 2006] Qin, Feng, Wang, Cheng, Li, Zhenmin, Kim, Ho-seop, Zhou, Yuanyuan, and Wu, Youfeng; "Lift: A low-overhead practical information flow tracking system for detecting security attacks"; *Microarchitecture, 2006. MICRO-39. 39th Annual IEEE/ACM International Symposium on*; IEEE; 2006, 135–148 (cit. on pp. x, 24).

[Rabiner and Juang 1986] Rabiner, Lawrence and Juang, B; "An introduction to hidden Markov models"; ieee assp magazine 3.1 (1986), 4–16 (cit. on p. 27).

[Ruwase et al. 2008] Ruwase, Olatunji, Gibbons, Phillip B, Mowry, Todd C, Ramachandran, Vijaya, Chen, Shimin, Kozuch, Michael, and Ryan, Michael; "Parallelizing dynamic information flow tracking"; *Proceedings of the twentieth annual symposium on Parallelism in algorithms and architectures*; ACM; 2008, 35–45 (cit. on p. 24).

[Schulist et al. 2016] Schulist, Jay, Borkmann, Daniel, and Starovoitov, Alexei; *Linux socket filtering aka Berkeley packet filter (BPF)*; 2016; URL: `https://www.kernel.org/doc/Documentation/networking/filter.txt` (cit. on p. 80).

[Schwarz 1978] Schwarz, Gideon; "Estimating the dimension of a model"; The annals of statistics 6.2 (1978), 461–464 (cit. on pp. 28, 59).

[Seshadri and Sundberg 1994] Seshadri, Nambirajan and Sundberg, Carl-Erik W; "List Viterbi decoding algorithms with applications"; Communications, IEEE Transactions on 42.234 (1994), 313–323 (cit. on p. 28).

[Sexton et al. 2015] Sexton, Joseph, Storlie, Curtis, and Neil, Joshua; "Attack chain detection"; Statistical Analysis and Data Mining: The ASA Data Science Journal 8.5-6 (2015), 353–363 (cit. on pp. viii, 20).

[Siaterlis et al. 2013] Siaterlis, Christos, Genge, Béla, and Hohenadel, Marc; "EPIC: a testbed for scientifically rigorous cyber-physical security experimentation"; Emerging Topics in Computing, IEEE Transactions on 1.2 (2013), 319–330 (cit. on p. 35).

[Siddiqi et al. 2007] Siddiqi, Sajid M, Gordon, Geoffrey J, and Moore, Andrew W; "Fast state discovery for HMM model selection and learning"; *International Conference on Artificial Intelligence and Statistics*; 2007, 492–499 (cit. on p. 29).

[Sommers et al. 2004] Sommers, Joel, Yegneswaran, Vinod, and Barford, Paul; "A framework for malicious workload generation"; *Proceedings of the 4th ACM SIGCOMM conference on Internet measurement*; ACM; 2004, 82–87 (cit. on pp. 36, 91).

[Sood and Enbody 2013] Sood, Aditya K and Enbody, Richard J; "Targeted cyberattacks: a superset of advanced persistent threats"; IEEE security & privacy 1.1 (2013), 54–61 (cit. on p. 12).

[Stamp 2004] Stamp, Mark; "A revealing introduction to hidden Markov models"; Department of Computer Science San Jose State University (2004) (cit. on pp. 26, 56).

[Suh et al. 2004] Suh, G Edward, Lee, Jae W, Zhang, David, and Devadas, Srinivas; "Secure program execution via dynamic information flow tracking"; *Acm Sigplan Notices*; vol. 39; 11; ACM; 2004, 85–96 (cit. on pp. x, 22–24).

[Sundaram 2000] Sundaram, Ramasubramanian H; *The Baum-Welch algorithm*; tech. rep.; Technical Report, 2000 (cit. on p. 28).

[Tankard 2011] Tankard, Colin; "Advanced Persistent threats and how to monitor and deter them"; Network security 2011.8 (2011), 16–19 (cit. on pp. 12, 13).

[Titman and Sharples 2008] Titman, Andrew C and Sharples, Linda D; "A general goodness-of-fit test for Markov and hidden Markov models"; Statistics in medicine 27.12 (2008), 2177–2195 (cit. on p. 30).

[H. Tong 1975] Tong, Howell; "Determination of the order of a Markov chain by Akaike's information criterion"; Journal of Applied Probability (1975), 488–497 (cit. on p. 28).

[Trend Micro 2013] Trend Micro; *SAFE: a targeted threat*; 2013; URL: `http://www.trendmicro.com/cloud-content/us/pdfs/security-intelligence/white-papers/wp-safe-a-targeted-threat.pdf` (cit. on pp. 12, 13, 83).

[Valeur 2006] Valeur, Fredrik; *Real-time intrusion detection alert correlation*; University of California, Santa Barbara, 2006 (cit. on p. 14).

[Vance 2014] Vance, Andrew; "Flow based analysis of Advanced Persistent Threats detecting targeted attacks in cloud computing"; *Infocommunications Science and Technology, 2014 First International Scientific-Practical Conference Problems of*; IEEE; 2014, 173–176 (cit. on pp. viii, 16).

[Vasilomanolakis 2016] Vasilomanolakis, Emmanouil; "On Collaborative Intrusion Detection"; PhD thesis; Technische Universität Darmstadt, 2016 (cit. on p. 35).

[Vasilomanolakis et al. 2016] Vasilomanolakis, Emmanouil, Cordero, Carlos Garcia, Milanov, Nikolay, and Mülhäuser, Max; "Towards the creation of synthetic, yet realistic, intrusion detection datasets"; *Network Operations and Management Symposium (NOMS), 2016 IEEE/IFIP*; 2016, 1209–1214 (cit. on p. 35).

[Verizon 2010] Verizon; *2010 Data Breach Investigations Report*; 2010; URL: `http://www.verizonenterprise.com/resources/reports/rp_2010-data-breach-report_en_xg.pdf` (cit. on p. 15).

[Virvilis et al. 2013] Virvilis, Nikos, Gritzalis, Dimitris, and Apostolopoulos, Theodoros; "Trusted Computing vs. Advanced Persistent Threats: Can a defender win this game?"; *Ubiquitous Intelligence and Computing, 2013 IEEE 10th International Conference on and 10th International Conference on Autonomic and Trusted Computing (UIC/ATC)*; IEEE; 2013, 396–403 (cit. on p. 16).

[Viterbi 1967] Viterbi, Andrew; "Error bounds for convolutional codes and an asymptotically optimum decoding algorithm"; IEEE transactions on Information Theory 13.2 (1967), 260–269 (cit. on p. 27).

[Y. Wang et al. 2014] Wang, Yuan, Wang, Yongjun, Liu, Jing, and Huang, Zhijian; "A Network Gene-Based Framework for Detecting Advanced Persistent Threats"; *P2P, Parallel, Grid, Cloud and Internet Computing (3PGCIC), 2014 Ninth International Conference on*; IEEE; 2014, 97–102 (cit. on p. 17).

[Websense 2011] Websense; "Advanced Persistent Threats and Other Advanced Attacks: Threat Analysis and Defense Strategies for SMB, Mid-size, and Enterprise Organizations"; *Whitepaper*; 2011 (cit. on p. 11).

[Ye et al. 2004] Ye, Nong, Zhang, Yebin, and Borror, Connie M; "Robustness of the Markov-chain model for cyber-attack detection"; Reliability, IEEE Transactions on 53.1 (2004), 116–123 (cit. on p. 31).

[Yin et al. 2007] Yin, Heng, Song, Dawn, Egele, Manuel, Kruegel, Christopher, and Kirda, Engin; "Panorama: capturing system-wide information flow for malware detection and analysis"; *Proceedings of the 14th ACM conference on Computer and communications security*; ACM; 2007, 116–127 (cit. on p. 24).

[Q. Zhang et al. 2009] Zhang, Qiang, Man, Dapeng, and Wu, Yang; "Using HMM for intent recognition in cyber security situation awareness"; *Knowledge Acquisition and Modeling, 2009. KAM'09. Second International Symposium on*; vol. 2; IEEE; 2009, 166–169 (cit. on pp. 32, 91).

## List of Tables



## List of Figures

Guillaume Brogi

# Real-time detection of Advanced Persistent Threats using Information Flow Tracking and Hidden Markov Models

**Abstract:**
In this thesis, we present the risks posed by Advanced Persistent Threats (APTs) and propose a two-step approach for recognising when detected attacks are part of one. This is part of the Akheros solution, a fully autonomous Intrusion Detection System (IDS) being developed in collaboration by three PhD students. The idea is to use machine learning to detect unexpected events and check if they present a security risk. The last part, and the subject of this thesis, is the highlighting of APTs. APT campaigns are particularly dangerous because they are performed by skilled attackers with a precise goal and time and money on their side.
We start with the results from the previous part of the Akheros IDS: a list of events, which can be translated to flows of information, with an indication for events found to be attacks. We find links between attacks using Information Flow Tracking. To do so, we create a new taint for each detected attack and propagate it. Whenever a taint is on the input of an event that is part of another attack, then the two attacks are linked. However, the links are only potential because the events used are not precise enough, which leads to erroneously propagated taints. In the case of an undetected attack, no taint is created for that attack, but the other taints are still propagated as normal so that previous attack is still linked to the next attack, only skipping the undetected one.
The second step of the approach is to filter out the erroneous links. To do so, we use a Hidden Markov Model to represent APTs and remove potential attack campaign that do not fit the model. This is possible because, while each APT is different, they all go through the same phases, which form the hidden states of our model. The visible observations are the kind of attacks performed during these phases. In addition, the results in one phase dictate what the attackers do next, which fits the Markov hypothesis. The score used to rank potential attack campaign from most likely an APT to least likely so is based on a customised Viterbi algorithm in order to take into account potentially undetected attacks.

**Keywords:**
Intrusion Detection System, Advanced Persistent Threat, Information Flow Tracking, Hidden Markov Model

**Résumé :**
Dans cette thèse, nous présentons les risques posés par les Menaces Persistantes Avancées (APTs) et proposons une approche en deux temps pour distinguer les attaques qui en font partie. Ce travail fait partie d'Akheros, un Système de Détection d'Intrusion (IDS) autonome développé par trois doctorants. L'idée est d'utiliser l'apprentissage machine pour détecter des évènements inattendus et vérifier s'ils posent un risque de sécurité. La dernière étape, et le sujet de cette thèse, est de mettre en évidence les APTs. Les campagnes d'APT sont particulièrement dangereuses car les attaquants sont compétents et ont un but précis ainsi que du temps et de l'argent.
Nous partons des résultats des parties précédentes d'Akheros : une liste d'évènements traduisibles en flux d'information et qui indique quand des attaques sont détectées. Nous faisons ressortir les liens entre attaques en utilisant le Suivi de Flux d'Information : nous ajoutons une nouvelle teinte pour chaque attaque. Lors de la propagation, si une teinte se trouve en amont d'un flux qui fait partie d'une attaque, alors les deux attaques sont liées. Certaines attaques se trouvent liées par erreur car les évènements que nous utilisons ne sont pas assez précis, d'où l'approche en deux temps. Dans le cas où certaines attaques ne sont pas détectées, la teinte de cette attaque n'est pas créée, cependant, les autres teintes sont propagées normalement, et l'attaque précédent l'attaque non détectée sera liée à l'attaque lui faisant suite.
Le deuxième temps de l'approche est de retirer les liens erronés. Nous utilisons un Modèle de Markov Caché pour représenter les APTs et retirons les campagnes qui ne suivent pas le modèle. Ceci fonctionne car les APTs, quoique toutes différentes, passent par les mêmes phases. Ces phases sont les états cachés du modèle. Les observations sont les types d'attaques effectuées pendant ces phases. De plus, les actions futures des attaquants dépendent des résultats de l'action en cours, ce qui satisfait l'hypothèse de Markov. Le score utilisé pour classer les campagnes potentielles de la plus proche d'une APT à la plus éloigné est basé sur l'algorithme de Viterbi qui est modifié pour prendre en compte les attaques non détectées potentielles.

**Mots clés :**
Système de Détection d'Intrusion, Attaque Persistente Avancée, Suivi de Flux d'Information, Modèle de Markov Caché