Netzob : un outil pour la rétro-conception de protocoles de communication

Georges Bossert[1,2], Frédéric Guihéry[2], and Guillaume Hiet[1]

[1] Équipe CIDRE, Supelec,
avenue de la Boulaie, 35576 Cesson Sévigné
prenom.nom(@)supelec.fr,
http://www.supelec.fr
[2] Laboratoire d'évaluation d'AMOSSYS,
4 bis allée du Bâtiment, 35000 Rennes
prenom.nom(@)amossys.fr,
http://www.amossys.fr

Résumé Dans cet article, nous présentons Netzob [3], un outil libre de rétro-conception semi-automatisée de protocoles de communication. Netzob est destiné à répondre à différents cas d'applications (analyse de sécurité, génération de trafic réaliste, interopérabilité, *etc.*) où la compréhension d'un protocole propriétaire ou non documenté est primordiale.
Netzob s'appuie principalement sur des algorithmes issus des domaines de la bio-informatique et de la théorie des automates. Il propose également un module de simulation de trafic, permettant ainsi la génération de flux de communication réalistes issus de l'inférence de protocoles dont la spécification est inconnue.

1 Introduction

Ces dernières années, le domaine de l'analyse de sécurité de systèmes ou de logiciels s'est étoffé de nouvelles approches et de nouveaux outils basés sur le *fuzzing*. Comparés aux techniques plus traditionnelles (analyses statique et dynamique de binaire, potentiellement combinées à l'analyse de code source), qui demandent des compétences pointues, des ressources et du temps, ces nouveaux outils offrent de multiples avantages : relative simplicité de mise en œuvre, approche semi-automatisée, acquisition rapide de résultats, etc.

Pour autant, l'expérience montre que, pour être réellement efficace, l'analyse de sécurité par *fuzzing* nécessite une bonne connaissance de la cible, et en particulier du protocole de communication qui permet de dialoguer avec celle-ci. Ce constat limite ainsi l'efficacité et la complétude des résultats obtenus lors de l'analyse de produits implémentant des protocoles propriétaires ou non documentés.

3. http://www.netzob.org

D'autre part, lors de l'analyse de produits de sécurité tels que des pare-feu ou des solutions de détection d'intrusion réseau (NIDS), l'évaluateur est régulièrement confronté au besoin de générer du trafic réaliste, afin de mesurer la pertinence et la fiabilité du produit testé (c'est-à-dire sa capacité à limiter les faux positifs et les faux négatifs). Cette opération est complexe à mettre en œuvre car elle requiert la caractérisation et le contrôle complet du trafic généré, ce qui nécessite une connaissance approfondie du protocole [6].

Au delà de ces deux besoins (analyse de la sécurité d'une implémentation et génération d'un trafic représentatif d'un protocole propriétaire pour l'évaluation), il existe également d'autres contextes pour lesquels la connaissance fine du protocole de communication est nécessaire. On retrouve notamment ce besoin en rétro-conception de protocoles pour réaliser le portage d'applications et de matériels propriétaires, par exemple pour le développement de pilotes destinés à assurer le fonctionnement de périphériques sur un système d'exploitation à l'origine non géré.

C'est pour répondre à ces nombreux besoins en rétro-conception de protocoles que Netzob a été développé. Cet outil se veut un cadre de travail [4] pour l'inférence de protocoles. Il soutient l'expert dans son travail de rétro-conception en proposant un ensemble de fonctionnalités semi-automatisées pour réduire le temps d'analyse et finalement améliorer la compréhension du protocole ciblé.

Dans cet article, nos contributions sont les suivantes :

1. nous proposons une approche, combinant des techniques d'apprentissage passives et actives, qui étend l'utilisation d'algorithmes issus du monde de la bio-informatique à la rétro-conception de protocoles inconnus. En particulier :
   - nous proposons une approche complète, permettant de modéliser puis de simuler un protocole en prenant en compte son vocabulaire et sa grammaire,
   - nous contextualisons l'inférence et la simulation de protocoles en fonction de l'environnement,
   - nous utilisons une extension stochastique des machines de Mealy pour la modélisation des protocoles à états les plus complexes ;
2. nous proposons un outil libre, déjà utilisé dans un contexte d'évaluation de sécurité et qui implémente l'ensemble des fonctionnalités présentées.

4. *Framework* en anglais

Le reste de cet article est organisé comme suit. Dans la section 2, nous présentons la notion de langage en lien avec la définition d'un protocole de communication et l'état de l'art concernant l'inférence de protocoles. Ensuite, dans la section 3, nous décrivons les solutions mises en œuvres dans Netzob pour la réalisation de l'inférence du vocabulaire et de la grammaire. Enfin, nous présentons l'outil implémentant nos travaux (section 4) et son application sur la rétro-conception de protocoles de communications (section 5).

2 État de l'art

2.1 Définition d'un protocole de communication

Intuitivement, un protocole de communication se définit comme l'ensemble des règles permettant à une ou plusieurs entités (ou acteurs) de communiquer. Nous nous intéressons dans nos travaux aux protocoles utilisés dans les réseaux informatique et de télécommunication, ou, par extension, entre plusieurs couches logicielles s'exécutant sur une même machine.

Dans le domaine des réseaux, les protocoles ont fait l'objet de nombreux travaux de normalisation, notamment ceux de l'ISO (modèle OSI [31,17] qui établit, entre autre, le principe des couches protocolaires) et de l'IETF (pile TCP/IP [7]). Pour autant, peu de travaux se sont attachés à donner une définition formelle et générique d'un protocole. Nous reprenons ici celle utilisé par Gerard Holzmann dans son ouvrage de référence *Design and Validation of Computer Protocols* [16]. Selon l'auteur, une spécification d'un protocole est constituée de cinq parties distinctes :

1. le **service** que doit fournir le protocole ;
2. les **hypothèses** sur l'environnement dans lequel le protocole est exécuté ;
3. le **vocabulaire** des messages utilisés pour implémenter le protocole ;
4. l'**encodage (format)** de chaque message du vocabulaire ;
5. les **règles de procédure** (établissement d'une connexion, acquittement, etc.) permettant d'assurer la cohérence des messages échangés.

L'auteur fait également remarquer que cette définition est similaire à celle d'un langage : le vocabulaire et le format des messages définissent la syntaxe, les règles de procédure définissent la grammaire et la spécification des services définissent la sémantique du langage associé au protocole.

Dans nos travaux, cette spécification est inconnue et nous cherchons à l'inférer à partir de messages observés et d'une implémentation logicielle ou matérielle du protocole. Comme le fait remarquer Holzmann, la cinquième partie est la plus difficile à concevoir et à vérifier. Dans notre cas, c'est également la plus difficile à inférer. En outre, cette définition générique est, en quelque sorte, « fractale » puisque chaque partie peut définir une hiérarchie d'éléments. Par exemple, le format des messages peut définir d'autres messages imbriqués (on retrouve la notion de couche protocolaire définie dans les normes citées précédemment).

Dans nos travaux, nous cherchons essentiellement à inférer les trois derniers éléments de la spécification. Par la suite, nous considérons que l'inférence protocolaire nécessite d'apprendre à la fois :

- l'ensemble des messages et leur format, que nous regroupons sous le terme de **vocabulaire** du protocole ;
- l'ensemble des règles de procédure que nous appelons **grammaire** du protocole.

Les sous-sections suivantes décrivent l'état de l'art des travaux ainsi que des outils existants permettant l'automatisation de l'inférence du vocabulaire (section 2.2) et de la grammaire (section 2.3) d'un protocole.

2.2 Inférence du vocabulaire d'un protocole

Les travaux de M. A. Beddoe [3] sont parmi les premiers à appliquer un algorithme issu de la bio-informatique à la rétro-conception automatique du vocabulaire d'un protocole. Dans le cadre de ses recherches, l'auteur publie notamment *The Protocol Informatic Project* (PI) [5], une application qui implémente un algorithme d'alignement de séquences nommé Needleman & Wunsch [22]. Cet outil automatise l'identification des champs composant chaque message en utilisant le résultat de l'alignement de plusieurs instances d'un même message. Il permet également le regroupement des messages similaires selon leurs alignements à l'aide d'un algorithme de *clustering* appelé UPGMA [6] [27].

En 2006, cette approche est réutilisée et étendue par W. Cui *et al.* [12] au travers de *RolePlayer*, un outil qui permet de générer un trafic issu d'un protocole inconnu. Pour cela, il identifie les champs composant les messages échangés selon la même approche que celle utilisée par M. A. Beddoe et détermine ensuite automatiquement les mutations à introduire pour obtenir un message différent mais toujours valide. Pour se faire, il

5. http://www.4tphi.net/~awalters/PI/PI.html
6. *Unweighted Pair Group Method with Arithmetic Mean*

ajuste la valeur des champs dits contextuels (contenant des adresses IP, des *cookies*, etc.).

Par la suite, J. Caballero *et al.* proposent une nouvelle approche qu'ils utilisent dans un outil nommé *Polyglot* [9]. Cette approche consiste à associer à l'analyse du trafic réseau, l'analyse de l'exécution d'un binaire participant à la communication pour en déduire le format des messages échangés. Leurs travaux sont ensuite étendus par ceux de G. Wondracek *et al.* [30] et ceux de Z. Lin *et al.* [20] qui corrèlent également le flux d'exécution d'un programme avec les messages échangés. Ces différentes solutions ont en commun d'être très efficaces si certaines conditions sur le contexte de l'inférence peuvent être respectées. En effet, cette approche ne peut s'appliquer que s'il est possible d'accéder au binaire et s'il est, en outre, possible de l'analyser lors de son exécution. Ceci n'est pas toujours évident, notamment lorsque l'on souhaite inférer le vocabulaire d'un logiciel malveillant protégé ou celui d'une application embarquée. En outre, l'identification du format des messages se base sur des intuitions de l'expert concernant les choix réalisés par le développeur du binaire (utilisation de constantes pour la représentation d'un délimiteur, présence de flux d'exécution équivalents pour la lecture de tous les champs optionnels d'un message, *etc.*).

Basé sur une autre approche, l'outil *Veritas* publié par W. Yipeng [28] permet l'apprentissage du format des messages en s'appuyant sur la présence de mots-clés identifiés de manière statistique. Dans un premier temps, *Veritas* détermine les mots-clés du langage les plus fréquents dans les messages capturés. Chaque mot-clé, dont la fréquence dépasse un seuil fixé au préalable, permet de définir un *cluster* comprenant l'ensemble des messages dans lesquels le mot-clé apparaît. Chaque *cluster* est ensuite caractérisé par une abstraction des messages. Cette approche suppose que chaque message puisse être caractérisé par un mot-clé unique et *vice-versa*. Toutefois, un même mot clé peut, en pratique, être identifié dans différents champs de différents messages, ce qui limite l'intérêt de cette approche. En outre, les auteurs fournissent peu de détails quant aux méthodes et aux outils statistiques utilisés.

Finalement, l'outil *Scriptgen* développé par C. Leita *et al.* [19] intègre des fonctionnalités liées à l'inférence de vocabulaire qui nous semblent être les plus avancées et les plus proches de celles intégrées dans Netzob. À l'origine conçu pour générer automatiquement des règles pour le pot de miel Honeyd [7], il réalise l'inférence du vocabulaire d'un protocole inconnu en optimisant les fonctionnalités de *PI* (l'outil proposé par Bed-

7. http://www.honeyd.org/

doe) et en proposant une extension sous la forme d'un algorithme nommé *Region Analysis algorithm*. Ce dernier consiste en une deuxième opération de regroupement, appelée « *micro-clustering* » sur les messages d'un même groupe pour en déduire d'autres sous-ensembles. Son objectif est de regrouper les messages ayant d'une part le même découpage en champs mais également des valeurs similaires pour certains champs.

Scriptgen a été développé dans l'objectif d'automatiser complètement l'inférence d'un vocabulaire. Nous pensons que cette approche purement automatique se heurte à un nombre important de limitations qui sont d'ailleurs identifiées par les auteurs. L'objectif de Netzob est, de ce point de vue, moins ambitieux. Notre outil doit aider l'expert en le laissant libre de modifier et d'adapter les résultats issus des phases d'apprentissage automatique. En outre, Netzob couvre un spectre fonctionnel plus large et permet également de réaliser l'inférence de la grammaire ainsi que la simulation des protocoles inférés. Enfin, Netzob est un outil libre, donc publiquement accessible, que les experts des différents domaines peuvent utiliser, tester, analyser et modifier s'ils le souhaitent.

Le domaine de l'inférence du vocabulaire, et plus spécifiquement celui de l'apprentissage du format des mots le composant, a fait l'objet de nombreux travaux. Cependant, sur l'ensemble des travaux présentés ci-dessus seuls ceux de Beddoe ont effectivement donné lieu à la réalisation d'un outil publiquement disponible. En pratique, ces travaux académiques sont peu (ou pas) utilisés par les ingénieurs qui réalisent la rétro-conception de protocoles, certainement du fait de l'absence d'outils publiquement disponibles mettant en œuvre ces approches, ce que soulignent certains experts de la rétro-conception [25,13]. Netzob a été conçu dans l'objectif de palier ce manque en proposant un outil semi-automatique et librement utilisable, tout en contribuant scientifiquement, avec de nouvelles approches, au domaine de l'inférence de protocoles dans ses différentes composantes.

2.3 Inférence de la grammaire d'un protocole

Comme expliqué dans la section 2.1, outre l'apprentissage du vocabulaire, la rétro-conception d'un protocole de communication nécessite également l'inférence de sa grammaire. Cette opération est une instance de l'« inférence inductive » [1]. Appliquée à nos travaux, il s'agit donc d'apprendre la grammaire d'un protocole inconnu à partir d'exemples qu'il faut généraliser.

Dans nos travaux, nous nous intéressons aux protocoles dont la spécification repose sur l'utilisation d'automates à états finis (FSM [8]), ce qui est le cas pour la plupart des protocoles utilisés couramment. Par définition, ces automates définissent des grammaires régulières. L'inférence grammaticale est aujourd'hui un domaine scientifique à part entière qui a fait, depuis une trentaine d'années, l'objet de nombreux travaux. Le lecteur intéressé par ce domaine pourra notamment consulter l'ouvrage de référence de Colin de la Higuera [15]. Bien qu'il s'agisse d'un domaine relativement fécond, peu de travaux se sont attachés à mettre en œuvre les approches et techniques de l'inférence grammaticale pour la rétro-conception de protocoles de communication.

De manière générale, le processus d'inférence fait appel à un « élève » à qui l'on donne accès à une source de données à partir de laquelle il extrait des phrases valides selon la grammaire inférée (dans notre cas, il pourra s'agir d'une liste des appels systèmes [11], de flux d'exécution d'un binaire [26], de paquets réseaux générés [28], etc.). Finalement, l'élève retourne une grammaire qui explique au mieux les données analysées.

Il existe deux types d'élèves : ceux qui analysent les échanges sans y participer [2,28,26,11]) et ceux qui, à l'inverse, participent aux échanges en communiquant avec une implémentation du langage [5,10]. En fonction du type d'élève utilisé, l'inférence est considérée comme passive dans le premier cas et active dans le second.

Inférence passive Dans le cas d'une inférence passive, l'élève est entièrement dépendant des communications observées puisque, par définition, il ne peut pas agir sur la génération des exemples en participant à la communication. Cette approche a l'avantage d'être simple et rapide à mettre en oeuvre puisqu'elle ne nécessite ni d'accéder, ni de communiquer avec une implémentation fonctionnelle de la cible analysée.

Parmi les différents travaux utilisant ce type d'élève, on retrouve notamment ceux de J. Antunes *et al.* [2] à l'origine de l'outil nommé *ReverX* [9] qui réalise automatiquement l'inférence d'un automate à partir des traces réseau d'un protocole inconnu. Son utilisation est réservée pour la rétro-conception de protocoles textes pour lesquels l'expert possède un ensemble de traces réseau. En outre, le modèle utilisé est relativement grossier.

Prospex, présenté dans [11] par P. Comparetti *et al.*, est un autre outil qui réalise passivement l'inférence de l'automate d'un protocole. Contrairement aux travaux de J. Antunes, il emploie un algorithme nommé *Ex-*

8. *Finite State Machine* en anglais
9. http://code.google.com/p/reverx

bar [18] pour inférer passivement la machine à états finis minimale décrivant un ensemble de traces. Celles-ci sont extraites lors d'une analyse de la mémoire d'un acteur d'une communication pendant son exécution. Le gain en précision apporté par cette approche nécessite la maîtrise complète du binaire ainsi que son instrumentation au sein d'un environnement maîtrisé.

Également basés sur un algorithme d'inférence passive, les travaux de S. Whalen *et al.*, présentés dans [29], modélisent une grammaire sous la forme d'un modèle de Markov caché. Ce modèle introduit une notion d'indéterminisme afin de répondre aux besoins de la détection en calculant les probabilités d'apparition pour chaque séquence de messages. Cette solution permet d'approximer le langage sans véritablement apprendre sa grammaire.

De la même manière, *Veritas* de W. Yipeng [28] n'utilise que des traces réseaux des couches applicatives pour inférer automatiquement la machine à états du protocole associé. En outre, la grammaire est modélisée par une machine à états probabiliste (P-PSM [10]). Celle-ci est déterminée en associant à chaque message un état et en calculant la probabilité que deux messages soient consécutifs. En plus d'être approximative, cette modélisation ne supporte pas l'utilisation d'un même message à deux endroits différents dans le graphe de transitions.

Ces différentes approches permettent d'inférer facilement la grammaire à partir d'exemples. Toutefois, la principale limite concerne la complétude et la précision du modèle inféré. C'est une limite inhérente aux approches passives qui justifie le recours aux approches actives. Plus précisément, deux facteurs limitent la complétude du modèle :

- le vocabulaire appris est un sous-ensemble du vocabulaire utilisé dans le langage cible ;
- la grammaire inférée à partir de ce vocabulaire appris est elle-même incomplète.

Les méthodes d'inférence active de grammaire supposent généralement que le vocabulaire soit connu au préalable. Dans Netzob, le vocabulaire est d'abord inféré de manière passive puis la grammaire est inférée de manière active à partir du vocabulaire appris. Nous espérons, de cette manière, réduire le deuxième facteur d'incomplétude.

Inférence active Par définition, la passivité de l'élève ne lui permet pas de découvrir les phrases qui ne sont pas couvertes explicitement par les

10. *Probabilistic Protocol State Machine* en anglais

observations réalisées. L'inférence active consiste donc à stimuler une implémentation en communiquant avec elle, de manière à assurer un meilleur contrôle sur les sources de données d'apprentissage. L'élève peut ainsi soumettre une série d'expériences à l'implémentation, aussi appelé « oracle » ou « professeur » en fonction des réponses qu'il apporte (réponse avec ou sans contre-exemple). Ces expériences permettent d'obtenir une connaissance plus complète du modèle. Cette approche peut être mise en œuvre pour inférer la grammaire du protocole mais, contrairement à l'inférence du vocabulaire d'un protocole, elle a fait l'objet d'un nombre plus réduit de travaux.

Cette méthode, notamment utilisée pour la génération automatique de cas de tests [14], a fait l'objet récemment de travaux appliqués à nos besoins. Parmi eux, les résultats publiés par Chian Yuan Cho *et al.* [10] démontrent la possibilité d'inférer activement une machine de Mealy [21] représentant un protocole de communication. Les auteurs utilisent l'algorithme L^*, proposé initialement par D. Angluin [1] et étendu par O. Niese aux automates à entrées sorties [23], qui détermine les séquences de messages les plus efficaces à soumettre à un oracle pour assurer la complétude et la précision nécessaire en un minimum d'expériences. Cependant, le modèle utilisé est relativement simple. Il ne prend pas en compte le temps de réaction entre la réception d'un message et l'émission de la réponse. En outre, il n'établit pas de contexte permettant la prise en compte les dépendances entre les champs de messages distincts.

T. Bohlin et B. Jonsson [5] ont étendu cette approche en représentant la grammaire inférée par un automate fini symbolique. Cette extension des FSM intègre la notion de symboles permettant d'abstraire plusieurs messages du même type. Dans ce modèle, le graphe de transition établissant la grammaire ne manipule donc pas directement les mots du vocabulaire mais les symboles correspondant. Cette solution permet de réduire la taille de l'automate inféré tout en préservant sa complétude. Cependant, aucune information n'est fournie par les auteurs concernant l'apprentissage des valeurs des symboles, ni même de la mise en place d'un contexte établissant l'évolution de leurs valeurs. Finalement, ce modèle ne considère ni le temps de réaction ni la possibilité d'avoir plusieurs messages de sortie pour le même couple état et message d'entrée.

Les principales limitations identifiées dans les travaux existants concernent essentiellement la source de donnée employée pour l'inférence ainsi que le modèle utilisé pour décrire une grammaire. En effet, l'évolution des protocoles informatique tend à rendre les grammaires de plus en plus complexes. Afin de répondre aux besoins liés à leur inférence, Net-

zob intègre un modèle évolué pour décrire de manière détaillée l'ensemble des séquences valides en considérant également le temps de réaction, la contextualisation des messages et l'indéterminisme en sortie.

À terme, nous envisageons également de profiter de l'inférence active de la grammaire pour éventuellement mettre à jour le vocabulaire appris au travers d'une boucle de rétroaction. Nous pourrions notamment utiliser des techniques proches du *fuzzing* pour améliorer la complétude de l'inférence du vocabulaire, mais cette approche n'est pas implémentée à l'heure actuelle dans Netzob.

3 L'inférence de protocoles dans Netzob

Ce chapitre présente la manière dont Netzob réalise l'inférence d'un protocole de communication dans son intégralité (vocabulaire et grammaire). Les objectifs, mécanismes et algorithmes sous-jacents à chaque étape sont expliqués, les aspects plus spécifiques à l'implémentation étant abordés dans la section 4.

Les travaux présentés dans cet article n'abordent pas les problématiques liées à l'identification d'un échange au sein d'un flux de données ni de son découpage en une série de messages. Ainsi, nous considérons que l'expert fournit en entrée de Netzob une liste de messages préalablement découpés. Toutefois, pour l'aider dans cette opération d'identification et de découpage d'un flux en messages, l'outil propose des modules spécifiques d'import (décrits plus en détail en section 4).

L'inférence d'un protocole est réalisée en trois principales étapes, représentées sur la figure 1, à partir des messages fournis en entrée par l'expert :

1. découpage des messages et regroupement par similarité ;
2. abstraction des messages regroupés afin d'obtenir un modèle plus général ;
3. inférence de la grammaire du protocole.

Bien que certaines de ces étapes soient entièrement automatisées, et d'autres purement manuelles, l'essentiel des fonctionnalités est d'ordre semi-automatique : l'outil fournit un résultat automatiquement et l'expert peut, par exemple, modifier les résultats obtenu ou ajuster certains paramètres des d'algorithmes. Le fonctionnement détaillé de chaque étape est présenté par la suite.

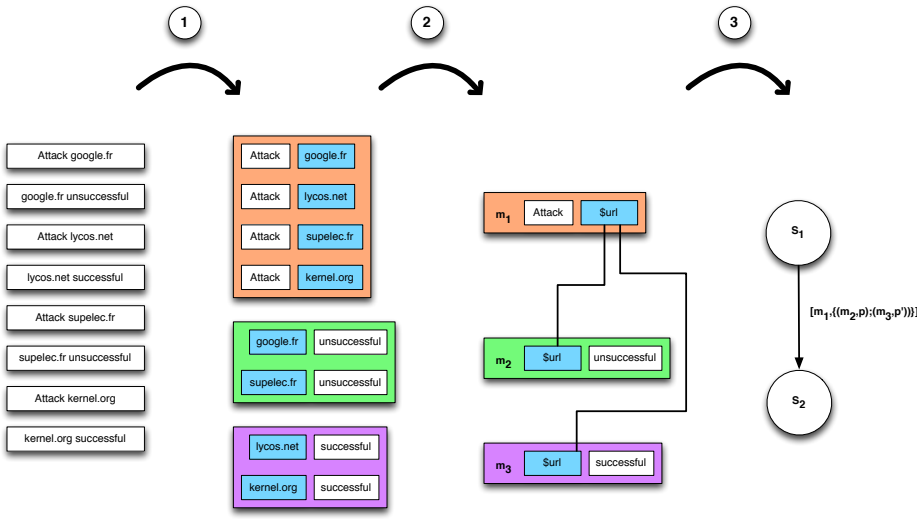

Figure 1. Les étapes de l'inférence de protocoles

3.1 Découpage des messages et regroupement par similarité

Dans Netzob, le vocabulaire est constitué d'un ensemble de **symboles**. Un symbole représente une abstraction d'un ensemble de messages similaires. Nous considérons que la notion de similarité se réfère à des messages ayant la même sémantique et/ou le même objectif du point de vue du protocole. Par exemple, l'ensemble des messages `TCP SYN` peut être abstrait par le même symbole. Une requête `ICMP ECHO REQUEST` et une commande `SMTP EHLO` sont d'autres exemples de symboles.

Un symbole est structuré suivant un **format**, qui spécifie un ensemble ou une séquence de **champs** autorisés. Par exemple, l'entête d'un paquet TCP et donc d'un symbole de son langage se compose de nombreux champs tels que ceux dédiés aux ports source et destination, aux sommes de contrôle et aux options.

Un champ peut lui-même être découpé en sous-parties (c'est par exemple le cas du champ *payload* de TCP). Ainsi, en considérant la notion de **couche protocolaire** comme un cas particulier de champ, il est possible de retrouver l'empilement protocolaire classique (TCP, encapsulé dans IP, lui même encapsulé dans Ethernet) à partir de simples messages ; chaque couche protocolaire ayant son propre vocabulaire et sa propre grammaire. L'identification des différents champs permet de définir le format des symboles (et donc des messages qu'ils représentent).

Une première difficulté se pose alors : comment identifier la délimitation d'un message en champs en étant suffisamment flexible pour couvrir la plupart des cas possibles (champs dynamiques à taille fixe, champs à taille variable avec indicateur de taille dans le message, champs à taille variable implicite, champs à taille variable séparés par un délimiteur, etc.) ?

Dans Netzob, la technique de l'alignement de séquences est utilisée pour inférer le format. Cette technique permet d'aligner les invariants sur un ensemble de messages. L'algorithme Needleman-Wunsh [22], bien connu du monde de la bio-informatique, remplit cette tâche de manière optimale. La figure 2 illustre, en première approximation, le découpage de quelques messages FTP grâce à ce principe d'alignement de séquences.

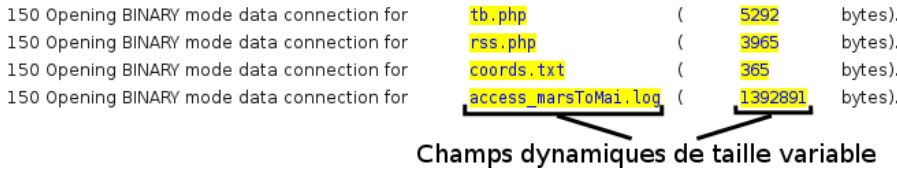

Figure 2. Alignement Needleman-Wunsh sur des messages FTP

L'algorithme de Needleman-Wunsh est particulièrement efficace sur des protocoles où les champs dynamiques sont de taille variable. Si la taille des champs est au contraire fixe, comme c'est le cas dans les entêtes DHCP ou TCP (sans les options), il est préférable d'utiliser un découpage simple. Un tel découpage est réalisé en séparant uniquement les données fixes des données variables. Un exemple d'application du découpage simple est proposé sur la figure 3, à partir du protocole DHCP.

Dans un cas typique de rétro-conception d'un protocole, il y a généralement un ensemble potentiellement important de messages comme données d'entrée. Il est nécessaire de pouvoir trier ces messages afin de les regrouper par similarité sémantique. L'algorithme de Needleman-Wunsch est donc utilisé en association avec un algorithme de regroupement (ou *clustering*). Comme dans la plupart des travaux, notamment ceux de Beddoe [3], celui retenu dans Netzob est l'UPGMA [27], également issu de la bio-informatique.

Cet algorithme utilise le résultat de l'alignement de séquence (matérialisé par une distance) pour construire une matrice contenant le résultat de l'alignement sur l'ensemble des couples de messages. Cette matrice de distances est ensuite transformée en un arbre enraciné (arbre phylo-

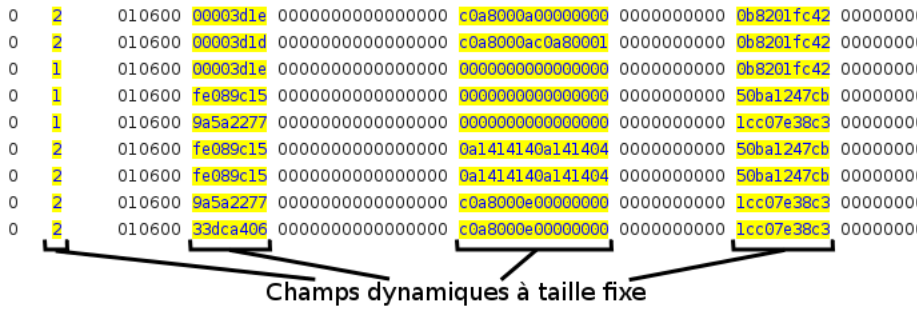

Figure 3. Découpage simple sur des messages DHCP

génétique), duquel est identifié le couple de messages (ou de groupes de messages) le mieux aligné. En réduisant progressivement la taille de la matrice, l'algorithme permet de déterminer les groupes de messages les mieux alignés, c'est-à-dire ceux ayant la meilleure similarité. À l'inverse, nous qualifions d'orphelin un message qui ne peut être aligné et donc regroupé avec d'autres (car son score de similarité est systématiquement trop faible).

L'efficacité du couple Needleman-UPGMA est optimale pour l'alignement et le regroupement de messages dont les champs statiques et les champs dynamiques sont répartis de manière homogène sur toute la longueur du message. À l'inverse, lorsque les messages comportent un ou plusieurs champs dynamiques dont la taille est prépondérante sur les champs statiques (typiquement, des champs d'entête), cette approche ne donne pas de résultats satisfaisants. C'est par exemple le cas pour des messages issus d'un protocole de type ASN-1. Comme l'illustre la figure 4, l'alignement est alors réalisé sur le contenu des champs dynamiques ce qui ne permet pas d'identifier des groupes de messages pertinents. Il en résulte souvent autant de groupes que de messages (beaucoup de messages sont « orphelins »).

Pour assurer la rétro-conception de ces familles de protocoles, nous avons étendu notre approche par l'ajout d'une heuristique que nous appelons *Orphan Reduction*. Le principe est le suivant : il s'agit d'appliquer le couple Needleman-UPGMA sur une partie du message sélectionnée par une fenêtre glissante. Ainsi, pour tous les messages orphelins dont l'alignement « traditionnel » a échoué, Netzob établit une fenêtre définissant la partie du message qui sera effectivement prise en compte pendant l'alignement et le regroupement.

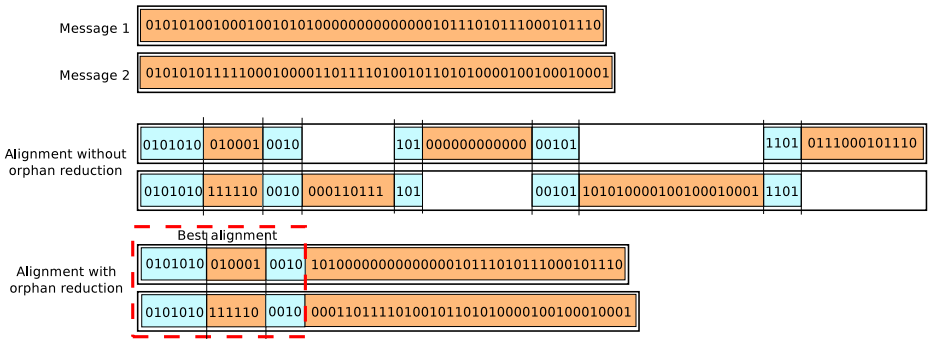

Figure 4. Comparaison de l'alignement avec et sans *Orphan Reduction* sur des messages de type ASN-1. Les deux messages représentés possèdent le même entête mais également une grande zone de données considérées comme aléatoires.

Comme illustré sur la figure 4, cette approche est d'abord utilisée pour aligner des messages dont l'entête fixe serait en début du message. Pour se faire, l'algorithme assigne tout d'abord la fenêtre à l'ensemble du message puis réduit progressivement la limite droite de la fenêtre et essaie, à chaque itération, d'aligner et de regrouper le message avec les groupes identifiés au préalable. Si à l'issue de cette opération, le message ne peut toujours pas être regroupé alors l'étape est réitérée en réduisant cette fois la fenêtre par la gauche. En se faisant, l'algorithme recherche une fenêtre optimale pour une partie fixe située en fin de message.

Notre approche pour regrouper les messages (avec ou sans l'étape d'*Orphan Reduction*) est uniquement basée sur le résultat de l'alignement de séquences. Cette approche fournit des résultats pertinents sur certaines familles de protocoles, notamment les protocoles (textes ou binaires) dont le format est variable d'un message à l'autre, mais n'est pas entièrement satisfaisante sur des protocoles binaires dont le format des messages est fixe. Des ajustements manuels complémentaires sont ainsi nécessaires pour optimiser le regroupement des messages. Un axe de travail est donc ouvert sur la détermination d'un meilleur facteur de similarité sémantique ou la mise en œuvre d'approches complémentaires.

À la suite des étapes d'alignement et de regroupement, nous obtenons une première approximation intéressante du format de chaque symbole. Il est ensuite possible de travailler au niveau des champs dans le but d'identifier leurs caractéristiques.

3.2 Abstraction des messages

Dans notre modèle, un champ possède différents attributs. Certains de ces attributs concernent tous les champs, d'autres sont fonction du type de champ et servent essentiellement pour la visualisation du contenu des champs. La visualisation correcte des données n'est pas essentielle pour les étapes automatiques, elle l'est en revanche pour celle nécessitant l'intervention de l'expert. Parmi ces attributs, seul le type est obligatoirement précisé (sa valeur par défaut est issue de l'étape précédente). Ces attributs sont les suivants :

- des attributs génériques, concernant tous les champs :
  - le **type** (obligatoire), qui représente le domaine de définition du champ, c'est-à-dire l'ensemble des valeurs autorisées (par exemple, un entier sur 16 bits, un booléen, une chaîne de caractères, etc.),
  - la **sémantique**, qui représente la signification du champ (par exemple, une adresse IP, un numéro de port, une URL, un *email*, un code d'intégrité, etc.) ;
- des attributs spécifiques :
  - la **structure**, dont la description peut être représentée dans une notation standard (ASN.1, TSN.1, EBML, etc.) ou *ad hoc*,
  - le **format de visualisation**, qui définit la manière dont doit être décodée la donnée brute (décimal, hexadécimal, ASCII, XML, EBML, DER, XER, PER, etc.),
  - le **boutisme (*endianness*)** qui spécifie l'orientation de l'interprétation des octets (orientation gros-boutiste ou petit-boutiste),
  - le **signe**, pour spécifier si une valeur numérique peut être signée ou uniquement positive.

Le type d'un champ est en partie issu de la phase d'identification du format des messages. Pour les champs contenant des invariants, le type correspond simplement à la valeur de l'invariant. Pour les autres champs, le type est matérialisé automatiquement, en première approximation, sous la forme d'une expression régulière, comme l'illustre la figure 5. Ce choix offre l'avantage de pouvoir valider facilement la conformité d'une donnée vis-à-vis du type attendu : il suffit de vérifier l'expression régulière sur la donnée. En complément, Netzob propose une fonctionnalité permettant de visualiser le domaine de définition d'un champ. De cette manière, l'expert peut raffiner manuellement le type associé à un champ.

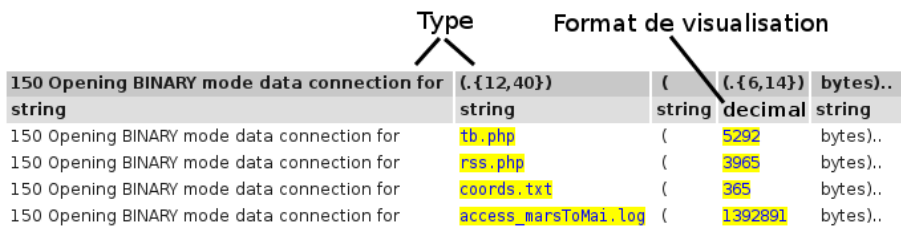

Figure 5. Abstraction des champs dynamiques sous la forme d'expressions régulières.

Les caractéristiques sémantiques sont recherchées à l'aide de motifs spécifiés au préalable : il s'agit d'expressions régulières [11] (*data carving*) ou de signatures caractéristiques (*file carving*). À titre d'exemple, l'expression régulière suivante est utilisée pour identifier les adresses IP encodées en ASCII :

```
(((?:25[0-5]|2[0-4][0-9]|[01]?[0-9][0-9]?)\.)
    {3}(?:25[0-5]|2[0-4][0-9]|[01]?[0-9][0-9]?))
```

Notons qu'il n'est pas systématiquement nécessaire de connaître la sémantique d'un champ. En effet, il est tout à fait envisageable de pouvoir générer du trafic en réutilisant des valeurs observées sans avoir à comprendre leur sens. Ceci est particulièrement vrai lorsque le champ est statique.

Il est possible de caractériser la valeur d'un champ de différentes manières. Un champ peut avoir :

- une valeur **statique** (par exemple, un nombre magique en entête d'un protocole) ;
- une valeur dépendant d'un autre champ, ou d'un ensemble de champs, du même symbole – notion de **dépendance intra-symbole** (par exemple, un code CRC) ;
- une valeur dépendant d'un champ, ou d'un ensemble de champs, d'un symbole transmis précédemment au sein d'une même session – notion de **dépendance inter-symboles** (par exemple, un numéro d'acquittement) ;
- une valeur dépendant de l'environnement – notion de **dépendance environnementale** (par exemple, une estampille temporelle) ;

11. Ces expressions régulières ne doivent pas être confondues avec celles identifiant les types des champs dynamiques.

– une valeur dépendant du comportement de l'application qui implémente le protocole – notion de **dépendance applicative** (par exemple, une URL rentrée par l'utilisateur) ;
– une valeur **aléatoire** (par exemple, la valeur initiale du numéro de séquence dans TCP).

Certaines dépendances intra-symbole sont identifiées automatiquement. C'est le cas du champ taille, présent dans de très nombreux protocoles, qui est un cas particulier de dépendance intra-symbole. Un algorithme a été développé dans le but d'identifier les potentiels champs taille et leur *payload* associée. Par extension, il est également possible d'identifier des protocoles encapsulés au sein d'un même message, en recherchant le champ spécifiant la taille de ces *payload*. A titre illustratif, la figure 6 montre une analyse de messages DNS complet, où il est possible d'identifier de manière automatique l'empilement protocolaire (UDP, encapsulé dans IP, lui même encapsulé dans Ethernet).

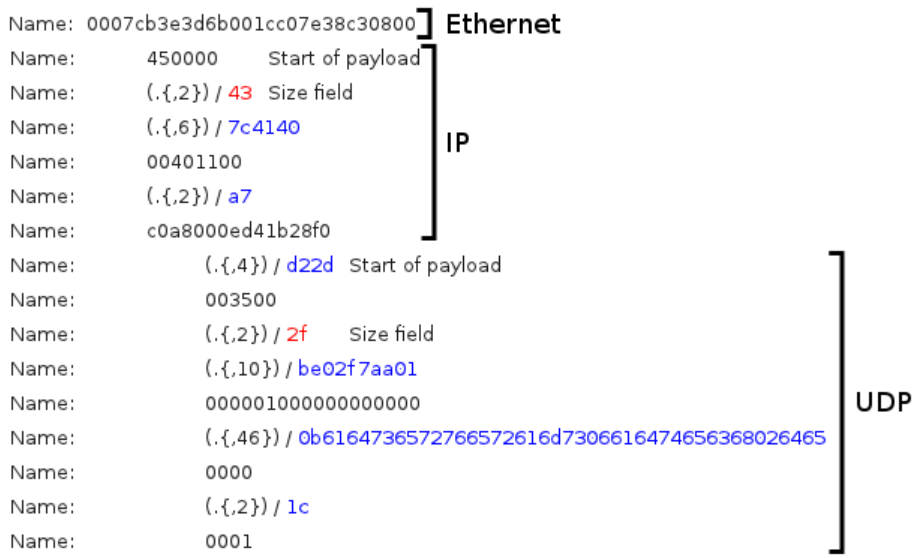

Figure 6. Extraction de protocoles encapsulés (ici, UDP dans IP dans Ethernet)

Les dépendances inter-symboles sont identifiées en recherchant des motifs identiques au sein de symboles successifs. Cette étape est pour l'instant manuelle.

Les dépendances environnementales sont identifiées en analysant les messages à la recherche d'éléments capturés lors de l'acquisition du trafic. Plusieurs caractéristiques propres à l'environnement (données liées au système d'exploitation, au matériel, à la configuration réseau, etc.) sont sauvegardées pendant cette phase. Lors de l'analyse, ces caractéristiques sont ensuite recherchées dans les messages suivant plusieurs formats d'encodage. Par exemple, lors de l'analyse de messages DHCP, il est possible d'identifier la sémantique de certains champs en recherchant des paramètres réseau spécifiques à l'environnement (adresse IP locale, adresse IP distante et adresse MAC), comme le montre la figure 7.

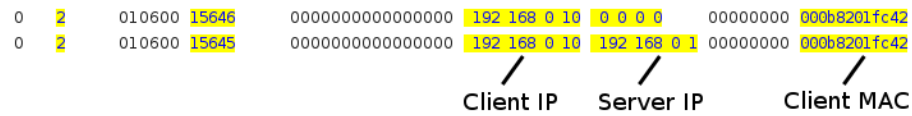

Figure 7. Identification de dépendances environnementales

Enfin, les dépendances applicatives sont identifiées en analysant les messages et les champs à la recherche de données spécifiques au comportement de l'application et de l'utilisateur. Cette étape est purement manuelle pour l'instant dans Netzob. Néanmoins, des possibilités d'automatisation sont envisageables. A titre d'exemple, il est tout à fait possible d'analyser les données écrites ou lues sur la sortie standard et de les rechercher dans les messages transmis. Cette voie sera explorée dans de futurs travaux.

En ce qui concerne l'identification des champs contenant des valeurs aléatoires, Netzob propose une analyse entropique afin de les mettre en évidence. De cette manière, il est possible d'identifier des clés cryptographiques de même que des champs contenant des messages chiffrés ou compressés.

La description de l'ensemble des ces relations qui spécifient la valeur d'un champ dynamique d'un symbole est réalisée grâce à l'emploi de **variables**. Ainsi le contenu d'un champ dynamique est représenté par une variable dédiée et qui regroupe l'ensemble des caractéristiques présentées précédemment. Par défaut, à l'issue de l'alignement et du regroupement, une variable est créée pour chaque champ dynamique et intègre les va-

leurs observées sous une forme normale disjonctive [12] telle qu'illustrée sur l'exemple de la figure 8.

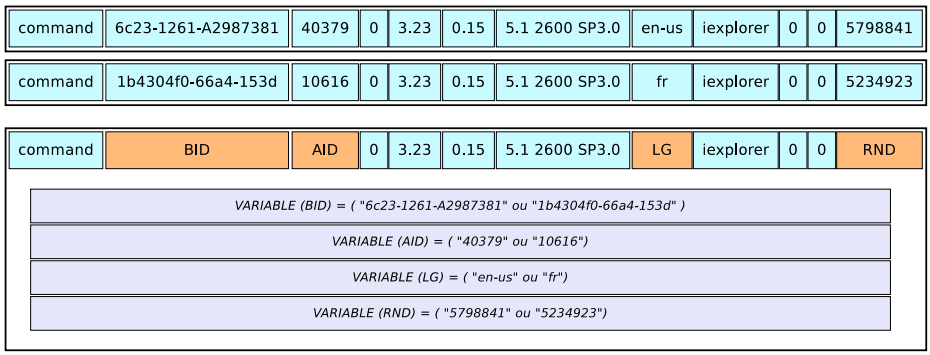

Figure 8. Illustration de la génération automatique d'une variable pour décrire la valeur des champs d'une requête générée par un bot TDL-4. Utilisation d'une forme normale disjonctive pour représenter les valeurs observées.

Une variable peut être définie par de simples motifs (adresse IP, suite de bits spécifique, *etc.*) mais également par des formes plus complexes combinant des agrégations et des alternatives de références sur d'autres variables. Il est également possible d'utiliser quelques opérations mathématiques simples telles que l'addition et la soustraction. A titre d'exemple, la figure 9 représente deux symboles utilisés dans la grammaire du protocole DHCP (entêtes BOOTP). On peut voir que le symbole `DHCP OFFER` se compose de plusieurs champs dynamiques associés à des variables. Certaines établissent un flux binaire aléatoire d'une taille donnée (champ 2 du `DHCP DISCOVER`). D'autres des références vers une autre variable (champs 2 et 6 du symbole `DHCP OFFER`).

3.3 Inférence de la grammaire

La grammaire d'un protocole spécifie les enchaînements valides de messages au sein d'une communication. Nous nous intéressons dans un premier temps aux grammaires régulières qui peuvent être modélisées par un automate à états finis (FSM). En effet, bien que ce modèle ne soit pas le plus expressif, il peut être inféré relativement facilement. Il est en outre

12. http://fr.wikipedia.org/wiki/Forme_normale_disjonctive

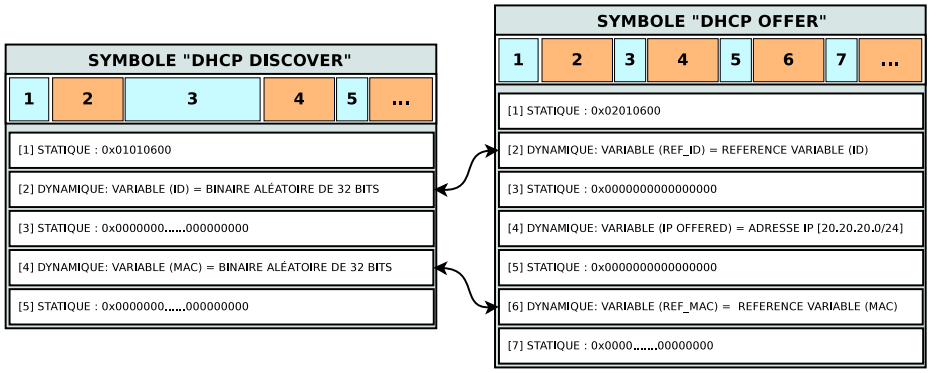

Figure 9. Exemple de deux symboles (`DHCP DISCOVER` et `DHCP OFFER`) utilisés par le modèle inféré du protocole DHCP.

souvent utilisé dans les spécifications de protocoles courants. Appliqués à la définition d'un protocole de communication, les états d'un automate traduisent différents contextes tandis que ses transitions représentent leurs évolutions en fonction d'événements (temps écoulé, message reçu, *etc.*).

Nous avons étendu et adapté un type de FSM existant afin de représenter au mieux les grammaires les plus complexes et permettre de palier l'absence de certaines informations contextuelles auxquelles l'outil ne peut avoir accès. En effet, Netzob n'a connaissance que des messages échangés et non de l'état complet de chaque participant, ni du contexte dans lequel il évolue. Par exemple, il ne peut inférer de manière déterministe la réponse à un message qui dépend du succès ou de l'échec d'une commande exécutée localement sur une des machines participant à la communication. Ces informations contextuelles, qui ne sont pas explicitement prise en compte dans le modèle, nécessitent de recourir à des modèles stochastiques, comme nous l'avons fait.

Notre modèle, que nous appelons **Machine de Mealy Stochastique à Transitions Déterministes** (MMSTD), est une extension des Machines de Mealy Stochastique (MMS) décrites par plusieurs auteurs [8,24], qui est elle même un automate à entrées sorties (Machine de Mealy [21]). Chaque transition de cet automate est étiquetée par :

- un symbole d'entrée (qui provoque l'exécution de la transition s'il est reçu par l'automate) ;
- un ou plusieurs symboles de sortie (associés à leurs probabilités respectives) qui sont émis lorsque la transition est effectuée ;

– une distribution des probabilités associée au temps d'émission des symboles de sortie (qui est une variable aléatoire).

Au sein d'une MMSTD, l'état courant évolue à la réception d'un message. À chaque réception d'un message, l'automate exécute la transition étiquetée par le symbole correspondant au message reçu. Si ce dernier ne correspond à aucun des symboles des transitions qui « partent » de l'état courant, l'automate ne change pas d'état et le message est « oublié ». L'exécution d'une transition a pour effet l'émission d'un message de sortie. Celui-ci correspond à un symbole de sortie associé à la transition, qui est tiré aléatoirement et qui est ensuite contextualisé (cela consiste à affecter une valeur aux variables associées à ses champs dynamiques). Ce message est ensuite émis après un temps d'émission qui est lui aussi choisi par un tirage aléatoire selon la loi de probabilité associée. Pour plus de détails sur la notion de MMSTD et sa description formelle, nous invitons le lecteur à se référer aux travaux que nous avons publiés dans un précédent article [6].

L'inférence d'une MMSTD et donc de la grammaire qu'elle décrit se divise en deux étapes. La première consiste à inférer une version déterministe de la MMSTD en utilisant une méthode active d'apprentissage. La seconde étape, passive, réalise la transformation de cet automate en MMSTD en comparant l'ensemble des séquences valides fournies en entrées de Netzob à celles décrites par l'automate.

Lors de la première étape, l'apprentissage actif consiste à communiquer avec une implémentation du protocole ciblé afin de réaliser des expériences. Les réponses fournies par cette implémentation permettent d'établir et d'améliorer une hypothèse (en fait, un automate) sur la grammaire du protocole utilisé. Finalement, lorsque que l'automate utilisé comme hypothèse est complet et cohérent (au sens de [4]), il est comparé avec celui de l'implémentation afin de le vérifier et, le cas échéant, de réitérer le processus.

L'efficacité de cette approche dépend donc essentiellement du nombre et de la qualité des expériences réalisées. Le choix de celles-ci est réalisé par une extension de l'algorithme L^* [1], qui s'applique à l'origine aux FSM. Cet algorithme est le plus efficace pour inférer le graphe de transition d'un modèle déterministe lorsque l'on dispose d'une implémentation du protocole. Afin de l'utiliser pour inférer une MMSTD, nous avons étendu cet algorithme afin de prendre en compte le temps de réaction et l'indéterminisme grâce à la seconde étape.

Pour appliquer l'algorithme L*, les deux pré-requis suivants doivent être vérifiés :

– le vocabulaire du protocole doit être connu au préalable ;
– il doit être possible de réinitialiser l'implémentation dans un état de référence (état initial).

Dans nos travaux, la première hypothèse est vérifiée grâce à l'étape d'inférence du vocabulaire. Bien évidemment, la qualité de l'inférence de la grammaire dépendra du taux de couverture du vocabulaire inféré qui dépend lui-même des traces capturées fournies en entrée. La deuxième hypothèse est vérifiée dès lors que l'implémentation est logicielle (il suffit, par exemple, d'utiliser des images disque ou des *snapshot* de machines virtuelles pour réinitialiser l'implémentation). Le cas des implémentations matérielles ou embarquées peut être plus problématique si ces implémentations n'offrent pas de fonction de réinitialisation.

Comme illustré au travers de la figure 10, le processus d'inférence implique un « élève » réalisant l'apprentissage de la grammaire d'une implémentation en soumettant des requêtes à un « oracle » et à un « professeur ». Une requête de sortie [13] consiste à stimuler l'« oracle » avec une séquence de symboles et d'observer sa sortie. Les symboles générés successivement par chacun des participants sont enregistrés et la relation entre chaque symbole émis et chaque symbole reçu est utilisée pour améliorer l'hypothèse sur le modèle.

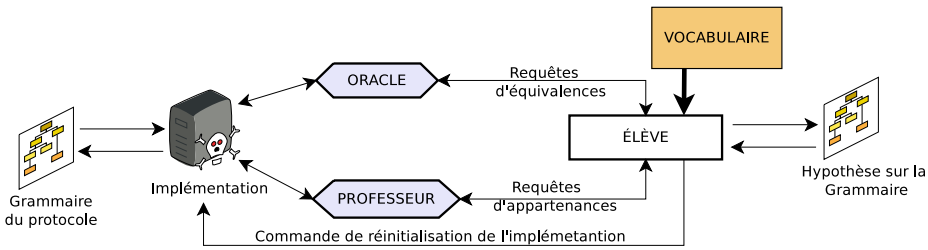

Figure 10. Inférence du graphe de transition d'un protocole à l'aide d'une extension d'Angluin.

La validation de l'hypothèse est réalisée grâce à l'exécution d'une requête d'équivalence [14]. Une telle requête consiste à rechercher un contre-exemple, c'est-à-dire une séquence de symboles valides dans l'implémentation et invalide dans l'hypothèse (ou *vice versa*).

13. en anglais, *Output Query*
14. en anglais, *Equivalence Query*

En réalité, nous ne soumettons pas le modèle à l'implémentation mais plutôt à un « professeur ». Ce dernier ne disposant pas du modèle « réel » (à inférer), il ne peut vérifier strictement l'équivalence de l'hypothèse à ce modèle. Il réalise donc une vérification par échantillonnage, assimilable à des tests de conformité. À partir de l'hypothèse à valider, il génère un ensemble de séquences de messages qu'il soumet ensuite à l'implémentation ainsi qu'au modèle inféré. La comparaison des séquences de sorties obtenues permet de déterminer si les deux modèles sont équivalents et donc si l'inférence est terminée. Si un contre-exemple est identifié, il est utilisé pour enrichir l'hypothèse et relancer le processus (reconstruction d'une hypothèse puis vérification approchée de l'équivalence). À l'inverse, si aucun contre-exemple ne peut être trouvé alors l'hypothèse est jugée valide. Bien entendu, il s'agit d'une approximation. La qualité du modèle inféré dépend donc également du taux de couverture des tests réalisés durant l'étape de vérification d'équivalence. Cette étape utilise un paramètre qui fixe implicitement le nombre de tests à réaliser et donc la durée d'apprentissage.

La deuxième étape consiste à modifier l'automate issu de l'étape précédente afin de le généraliser. En effet, l'inférence active permet d'apprendre le comportement spécifique d'un acteur (l'implémentation avec laquelle est réalisée l'inférence) utilisant le protocole à inférer. Les éléments purement contextuels sont également appris (adresse IP de la machine, nom d'utilisateur, réponse fonction de l'état interne de la machine, etc.). Afin de limiter les effets de ce sur-apprentissage, la deuxième étape utilise l'ensemble des traces fournit lors de l'inférence du vocabulaire (voir section 3.1) pour généraliser l'automate. Cette étape supplémentaire d'inférence passive permettra d'obtenir un modèle suffisamment générique si l'ensemble des traces est suffisamment hétérogène, c'est-à-dire si les traces proviennent d'échanges entre différents acteurs, appartenant à différent réseaux, utilisant des systèmes différents, etc.

Nous faisons l'hypothèse que ces variations contextuelles n'affectent que les symboles de sortie et le temps de réponse. Concrètement, cette seconde étape consiste à rejouer les traces sur l'automate inféré lors de l'étape précédente et, en comparant les symboles de sortie de l'automate aux messages de réponse présents dans les traces, à estimer, pour chaque état, les probabilités des différents symboles possibles. Cette seconde étape

permet également d'inférer la moyenne et la variance du temps d'émission des messages [15].

4 Implémentation de Netzob

Ce chapitre présente l'implémentation de Netzob. Sont notamment abordées les problématiques d'acquisition de données, de performance des algorithmes, de même que les choix retenus pour permettre la simulation de trafic.

Netzob est distribué sous licence GPLv3. À la date d'écriture de cet article (16 février 2012), le code source est constitué de 14000 lignes de code, essentiellement en Python. Nous avons réimplémenté l'ensemble des algorithmes utilisés car à notre connaissance, aucune version python et libre n'était disponible. En outre, certains algorithmes, dont le temps d'exécution est crucial, ont directement été développés en C et parrallélisés à l'aide de *pragmas* OpenMP [16].

Le code source est publiquement accessible sur un dépôt *git* [17] tout en étant également disponible en téléchargement sous la forme d'une archive *tar.gz* [18]. En outre, des paquets sont disponibles pour les distributions Linux Debian et Gentoo, de même que pour Windows. Netzob fonctionne sur les architectures x86 et x64.

L'interface graphique de Netzob est développée en PyGTK et repose sur le module matplotlib [19] pour la génération de graphiques, ainsi que sur le module Graphviz [20] pour la visualisation des automates de la grammaire.

4.1 Architecture fonctionnelle

La figure 11 présente l'architecture fonctionnelle de Netzob.

Les principaux modules de Netzob sont les suivants :

– **Module d'import de données**. L'import de données est réalisable de deux manières : soit en utilisant un des capteurs disponibles, soit en utilisant un format d'interface en XML.

15. Ce temps d'émission n'est pas toujours facilement mesurable. Il nécessite une capture au plus près de la cible. A défaut, il est nécessaire d'estimer le temps de parcours des messages entre la cible et la sonde.
16. Voir : http://openmp.org/wp
17. Voir : http://www.netzob.org/dev
18. Voir : http://www.netzob.org/download
19. Voir : http://matplotlib.sourceforge.net
20. Voir : http://www.graphviz.org

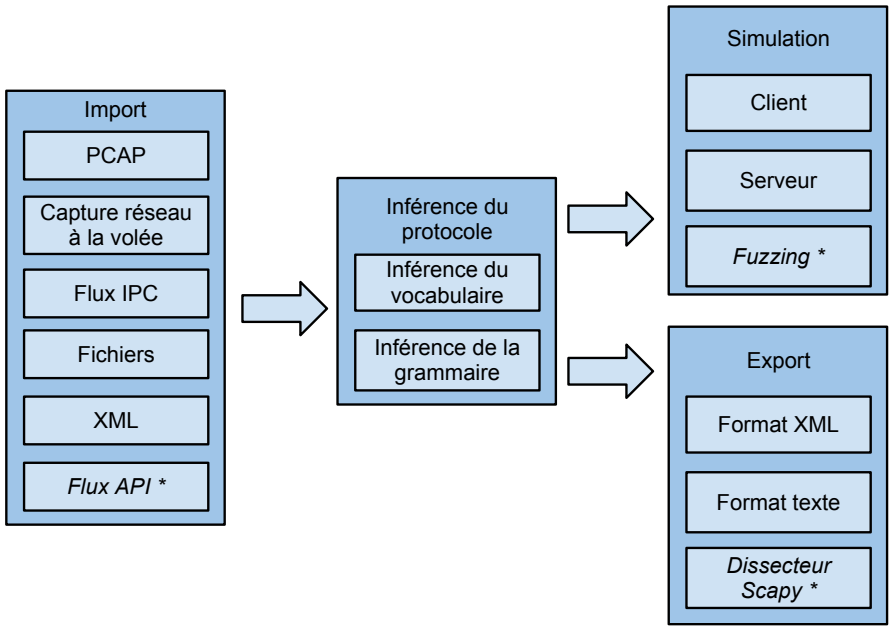

Figure 11. Architecture fonctionnelle de Netzob (les éléments marqués d'une astérisque sont en cours d'implémentation.)

- **Modules d'inférence de protocoles**. Les fonctions d'inférence du vocabulaire et de la grammaire d'un protocole constituent la partie centrale de Netzob . Ces fonctions ont été décrites en détails dans la section 3.
- **Module de simulation**. Une fois le protocole inféré, Netzob permet de générer du trafic respectant le modèle inféré du protocole. Le module de simulation autorise la création d'une instance d'un client ou d'un serveur dialoguant avec une implémentation réelle. Ce module de simulation est en cours d'extension pour permettre de réaliser du *fuzzing* sur le trafic généré ou observé.
- **Module d'export**. Le module d'export offre la possibilité d'exporter le modèle inféré d'un protocole dans des formats réutilisables par des applications tierces ou facilement compréhensibles par un humain. Le principal format d'export est matérialisé sous la forme d'un schéma XSD. Des travaux sont en cours pour permettre l'export des modèles de protocoles sous la forme de *dissector* pour Scapy [21] et

21. Voir : http://www.secdev.org/projects/scapy/

Wireshark [22], ainsi que dans des formats compatibles avec les *fuzzer* Peach [23] et Sulley [24].

Par la suite, nous présentons de manière plus approfondie les modules d'import de données (section 4.2) et de simulation de trafic (section 4.4). Nous abordons également les performances de l'inférence (section 4.3).

4.2 Import de données

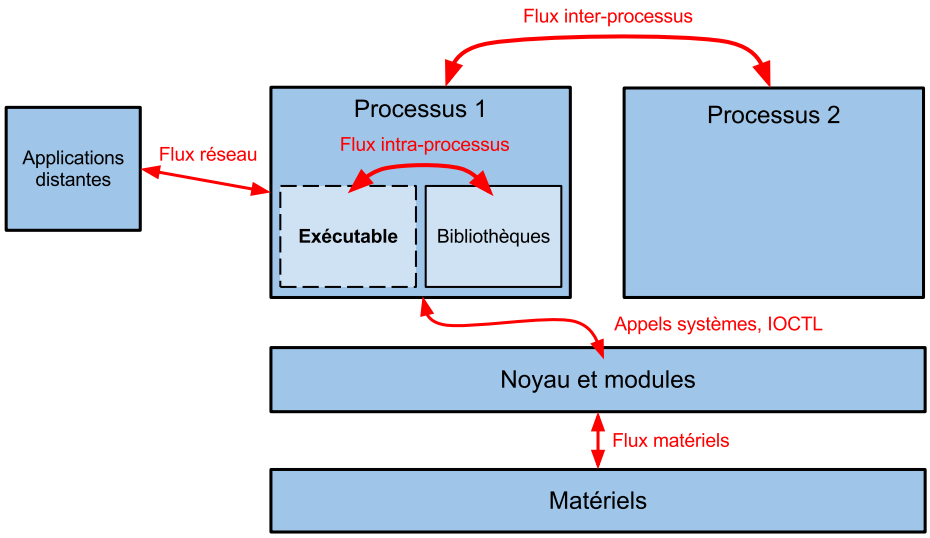

Figure 12. Omniprésence des protocoles de communication

Étant donné l'omniprésence des protocoles (voir figure 12) à tous les niveaux d'un système d'information, il est primordial de pouvoir offrir des possibilités de capture s'adaptant à un maximum de contextes. Trois approches sont retenues dans Netzob pour remplir cet objectif :

- la support natif de plusieurs types de capteurs ;
- la possibilité d'importer des données capturées dans un autre contexte (fichier PCAP ou simples fichiers de données) ;
- la possibilité d'intégrer facilement un nouveau capteur dans Netzob.

Actuellement, les sources de flux actuellement supportées par Netzob, au travers de modules de capture dédiés, sont les suivantes :

22. Voir : http://www.wireshark.org
23. Voir : http://peachfuzzer.com
24. Voir : http://code.google.com/p/sulley/

- **Flux réseau**. Les modules Python-dpkt [25] et Python-pcapy [26] sont utilisés afin de réaliser la capture de protocoles encapsulés dans TCP et UDP.
- **Traces réseau PCAP**. Netzob propose l'import de traces réseau sous la forme de fichiers au format PCAP. Cette fonction est particulièrement utile pour analyser des protocoles utilisés sur des systèmes non supportés par Netzob.
- **Flux inter-processus (IPC)**. La version pour Linux de Netzob propose la capture à la volée des données transférées entre processus. Il est notamment possible de filtrer les données à acquérir en fonction du type de canal (tube, mémoire partagée, *socket* réseau ou *socket* Unix) et du descripteur de fichiers. Netzob repose sur les utilitaires *strace* et *lsof* sous Linux pour la capture des flux IPC.
- **Fichiers structurés**. Netzob permet d'importer des données provenant de fichiers texte ou binaire dont la structure n'est pas connue. Ces fichiers peuvent être constitués de plusieurs messages séparés par un délimiteur qu'il est possible de spécifier pendant l'import.

En outre, Netzob est suffisament flexible pour autoriser la rétro-conception d'un protocole dont les flux sont capturés depuis des sources hétérogènes (par exemple, des fichiers structurés et du trafic réseau).

Lorsque les données proviennent de sources actuellement non supportées par les capteurs de Netzob, il est possible d'utiliser un format d'interface en XML. Ce format, spécifié sous la forme d'un schéma XSD, est également destiné à faciliter le développement de nouveaux capteurs (potentiellement décentralisés).

4.3 Performances de l'inférence du format des messages

La pertinence des résultats de l'inférence du format des messages dépend en grande partie de la qualité et de la quantité des sources de données. De fait, il est important de pouvoir mesurer les performances de Netzob face à des quantités importantes de messages. Ce chapitre fournit une analyse comparative des performances des algorithmes d'alignement de séquences (Needleman-Wunsh) et de regroupement (UPGMA), en fonction d'options de compilation. Les modes de compilation retenus sont :

- gcc sans aucune option d'optimisation ;
- gcc en mode d'optimisation -O3 ;
- gcc en mode d'optimisation -O3 avec l'intégration basique d'OpenMP (sur deux cœurs).

25. Voir :http://code.google.com/p/dpkt/
26. Voir : http://oss.coresecurity.com/projects/pcapy.html

La parallélisation OpenMP est intégrée au niveau de l'algorithme de regroupement. Chaque cœur du processeur traite le calcul d'un sous-ensemble distinct de la matrice des distances de UPGMA.

Le graphique 13 présente les résultats du comparatif sur différents protocoles (HTTP simple et basique, protocole P2P et données aléatoires). Les résultats correspondent au temps de calcul en secondes. Les traces utilisées pour ce test contiennent entre 20 et 50 messages dont la taille varie entre 50 et 500 octets.

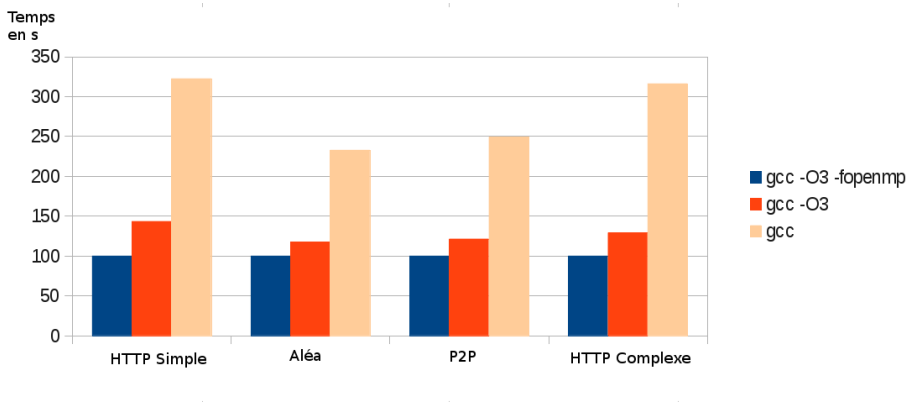

FIGURE 13. Performances de l'inférence du format des messages

Le premier constat que l'on peut faire est que le temps de calcul est relativement important pour de si faibles quantités de messages. Ce comportement est lié au fait que chaque trace contient plusieurs messages de taille importante, augmentant ainsi significativement le temps de calcul de chaque alignement de séquence.

Concernant les différentes options de compilation utilisées, en fonction du type de protocole, on peut observer une amélioration du temps de calcul allant de 17% jusqu'à 43% entre une optimisation classique (-O3) et le mode parallélisé avec OpenMP.

Ces observations montrent l'importance de l'optimisation des algorithmes Needleman-Wunsh et UPGMA dans l'objectif de pouvoir traiter un maximum de messages. Cet objectif est un pré-requis important si l'on veut obtenir une meilleure précision dans l'identification du format des messages.

Dans de futurs travaux, il est prévu d'améliorer la parallélisation des algorithmes et de porter les calculs à la fois sur des architectures distribuées (utilisation de la bibliothèque MPI), et éventuellement sur des cartes graphiques (technologie CUDA par exemple).

4.4 Simulation de trafic

Comme rappelé en introduction (voir section 1), la simulation de trafic réaliste, notamment à des fins d'évaluation de produits de sécurité, est un des besoins qui a justifié le développement de Netzob. En outre, et comme détaillé dans la section 3.3, la solution mise en place dans Netzob pour réaliser l'inférence de la grammaire d'un protocole nécessite la création de requêtes de sortie et donc la génération de trafic. Ces deux raisons ont donc justifié la création d'un module spécifique permettant de simuler un acteur (client, serveur, etc.) capable d'émettre et de recevoir des messages en respectant le protocole inféré. La figure 14 représente l'architecture retenue pour répondre à ce besoin.

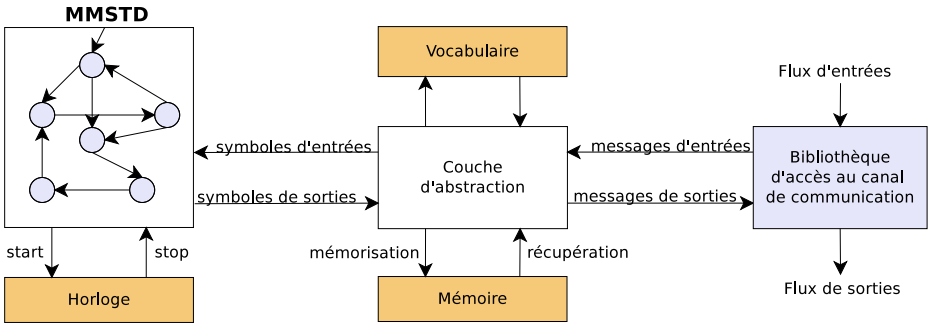

Figure 14. Architecture du générateur de trafic

En premier lieu, le simulateur utilise une bibliothèque dédiée au support utilisé pour la lecture et l'écriture sur le canal de communication. Celle-ci assure également la transformation du flux de données en messages et *vice-versa* lors de l'écriture. Cette bibliothèque étant spécifique à chaque canal, Netzob ne propose à l'heure actuelle que la simulation de communications réseaux applicatives, c'est-à-dire au dessus des protocoles de transport UDP ou TCP (utilisation de *sockets* réseaux). Netzob permet de spécifier si l'acteur simulé joue le rôle d'un serveur (en écoute)

ou d'un client (qui initie la connexion). Étant donné la modularité de l'architecture utilisée, nous envisageons à terme de proposer d'autres canaux de communications (USB, fichiers, IPC, etc.) grâce au développement de modules spécifiques.

Le second module est une « couche d'abstraction » qui permet la transformation d'un message d'entrée en symbole et à l'inverse la spécialisation d'un symbole en message de sortie. En utilisant les définitions des symboles stockées dans le vocabulaire ainsi qu'une mémoire pour gérer les variables inter-symboles, ce module permet de contextualiser les symboles manipulés par la MMSTD.

Finalement, le dernier module gère l'état courant de l'acteur simulé. Il s'appuie pour cela sur la définition de la grammaire du protocole (la MMSTD). Il fait évoluer l'automate en fonction des symboles d'entrée reçus. Le temps de réaction est quant à lui mesuré et simulé grâce à une horloge déclenchée à chaque transition.

5 Exemples d'inférence de protocoles

Nous présentons par la suite deux exemples illustrant l'apprentissage du vocabulaire d'un protocole standard (DHCP) et l'inférence de la grammaire utilisée sur le C&C d'un *botnet*.

5.1 Apprentissage du vocabulaire d'un protocole standard (DHCP)

Afin d'illustrer la capacité de Netzob à inférer le vocabulaire, nous avons appliqué l'outil sur un protocole standard connu : DHCP. La première étape de cette expérimentation est de réaliser l'inférence automatique du format des symboles de DHCP. Sans connaissance *a priori* du format des symboles, nous appliquons successivement le découpage par alignement de séquences, puis le découpage simple. Il s'avère que ce dernier produit un résultat plus satisfaisant sur la première partie des messages (on identifie nettement les champs, comme le montre la figure 15), tandis que le découpage par alignement de séquences s'applique mieux sur la fin des messages DHCP. De cette première étape, on peut en déduire que le protocole DHCP est constitué d'un entête à champs de taille fixe, puis d'une *payload* à champs de taille variable.

Dans une seconde étape, nous essayons de découvrir quelques caractéristiques au sujet des différents champs. Une analyse des dépendances environnementales permet d'identifier rapidement la présence de l'adresse

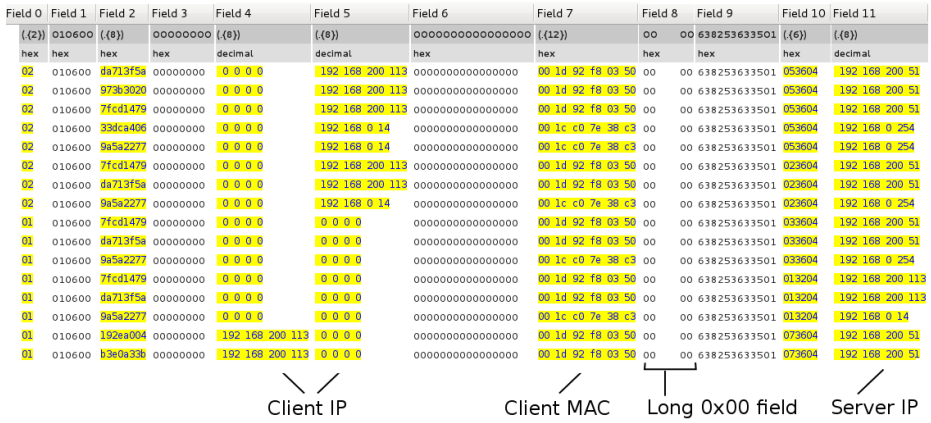

FIGURE 15. Première approximation du format des symboles DHCP.

IP du client et du serveur, de même que l'adresse MAC du client. L'utilisateur peut alors modifier le format de visualisation de ces champs afin de proposer un mode d'affichage plus classique pour les adresses IP (sous la forme de décimales sur un octet) et les adresses MAC (sous la forme hexadécimale sur un octet).

Une analyse manuelle complémentaire permet d'identifier rapidement les dépendances associées aux deux premiers champs dynamiques. La valeur du premier champ dynamique est liée à la direction de la transmission du message. La valeur est égale à 1 dans le sens client ↔ server et elle est égale à 2 dans le sens serveur ↔ client. Pour le deuxième champ dynamique, il s'avère que la valeur est constante au sein d'une même session DHCP. On peut donc supposer qu'il s'agit d'un champ d'identification de la session. Des travaux sont en cours pour permettre l'identification automatique des dépendances liées aux sessions et à la direction de la transmission des messages.

L'analyse manuelle des dépendances applicatives permet d'identifier aisément d'autres champs : l'adresse IP du routeur, le domaine DNS (facilement identifiable en « jouant » sur le mode de visualisation, car présent sous la forme d'une chaîne de caractères), l'adresse IP du serveur DNS et le masque de sous-réseau. Ces différents champs, non représentés sur la figure, apparaissent dans la partie identifiée comme la *payload* des symboles DHCP.

Enfin, le champ 10 sur la figure 15 varie à chaque étape d'une session DHCP, mais on retrouve ses valeurs d'une session à l'autre. Cela semble

indiquer un type de message au sein d'une session DHCP. Sur l'ensemble des traces analysées, nous pouvons identifier 6 types de messages différents.

A la fin de cette inférence semi-automatisée du vocabulaire, nous obtenons ainsi une approximation satisfaisante du format des principaux symboles du protocole DHCP, qui correspondent à *Discover*, *Offer*, *Request*, *Ack*, *Nack* et *Release*. Il s'agit d'une approximation car certains champs dynamiques sont en réalité composés de plusieurs champs dynamiques contigus, si l'on se réfère à la spécification RFC du protocole. A titre illustratif, si l'on essaie de modéliser entièrement un symbole DHCP du type *Request* avec Netzob, nous obtenons une sortie équivalente à celle produite par Wireshark (voir figure 16).

Wireshark output

```
Bootstrap Protocol
  Message type: Boot Request (1)
  Hardware type: Ethernet
  Hardware address length: 6
  Hops: 0
  Transaction ID: 0xda713f5a
  Seconds elapsed: 0
▷ Bootp flags: 0x0000 (Unicast)
  Client IP address: 0.0.0.0 (0.0.0.0)
  Your (client) IP address: 0.0.0.0 (0.0.0.0)
  Next server IP address: 0.0.0.0 (0.0.0.0)
  Relay agent IP address: 0.0.0.0 (0.0.0.0)
  Client MAC address: Micro-St_f8:03:50 (00:1d:92:f8:03:50)
  Client hardware address padding: 00000000000000000000
  Server host name not given
  Boot file name not given
  Magic cookie: DHCP
▽ Option: (t=53,l=1) DHCP Message Type = DHCP Request
    Option: (53) DHCP Message Type
    Length: 1
    Value: 03
▽ Option: (t=54,l=4) DHCP Server Identifier = 192.168.200.51
    Option: (54) DHCP Server Identifier
    Length: 4
    Value: c0a8c833
▽ Option: (t=50,l=4) Requested IP Address = 192.168.200.113
    Option: (50) Requested IP Address
    Length: 4
    Value: c0a8c871
▽ Option: (t=55,l=13) Parameter Request List
    Option: (55) Parameter Request List
    Length: 13
    Value: 011c02030f06770c2c2f1a792a
    1 = Subnet Mask
    28 = Broadcast Address
    2 = Time Offset
    3 = Router
```

Netzob output

```
<symbol name="Request">
  OpCode         : 0x01
  HwType         : 0x01
  Hw Addr Len    : 0x06
  Hops           : 0x00
  Serial         : (.{8})
  Seconds        : 0x0000
  Flags          : 0x0000
  Client IP      : 0x00000000
  Assigned IP    : 0x00000000
  Server IP      : 0x00000000
  Gw IP          : 0x00000000
  Client MAC     : (.{12})
  Boot Filename  : 0x0000000000000...0000

  Magic Number   : 0x63825363

  Message Type   : 0x35
  Len            : 0x01
  Message Value  : 0x03

  Server Identifier : 0x36
  Len            : 0x04
  Server IP      : (.{8})

  Requested IP   : 0x32
  Len            : 0x04
  IP Value       : (.{8})

  Parameter List Request : 0x37
  Len            : 0x0d
  Parameter List Value : 0x011c020...792a

  Option End     : 0xff
  Padding        : 0x00000000000000...0000
</symbol>
```

Figure 16. Exemples de rendu du format d'un symbole *DHCP Request*, dans Wireshark et Netzob

La section suivante présente la manière dont peut être inférée la grammaire d'un protocole inconnu utilisé sur la canal de communication d'un *botnet*.

5.2 Apprentissage de la grammaire d'un protocole inconnu

Dans cette section, nous détaillons la procédure d'apprentissage de la grammaire utilisée par un *botnet* sur son canal de communication (C&C). En l'occurrence nous considérons le *malware* associé au bot *PHP/Bot* aussi appelé *pbot*. En prérequis, nous considérons que l'expert a déjà réalisé l'apprentissage du vocabulaire utilisé par le *malware* suivant une approche équivalente à celle présentée pour le DHCP.

Pour inférer la grammaire, nous avons volontairement infecté un serveur web déployé dans une machine virtuelle. Celle-ci a ensuite été confinée au sein d'un réseau d'expérimentation destiné à analyser et intercepter l'ensemble des communications issues du C&C du malware.

Pour inférer ce protocole, la première étape pour l'expert consiste à rédiger un « script de réinitialisation » afin de réinitialiser la machine virtuelle en la faisant revenir à son état initial par l'intermédiaire d'un *snapshot*. Pour se faire, nous utilisons l'utilitaire *vboxmanage* qui permet de gérer des images *VirtualBox*.

La configuration de l'inférence de la grammaire est réalisée grâce à l'interface graphique de Netzob. Avant de lancer le processus d'inférence, l'expert doit fournir les informations suivantes :

- les paramètres du support de communication (port source TCP, adresse IP destination, *etc.*) ;
- l'emplacement du script de réinitialisation ;
- une estimation du nombre maximum d'états du protocole afin d'optimiser le temps de calcul.

Au cours de la procédure d'inférence, l'utilisateur peut suivre l'évolution de l'apprentissage grâce à une fenêtre de suivi. Celle-ci est mise à jour régulièrement à l'aide des données obtenues lors des expériences les plus récentes réalisées sur l'implémentation. L'interface affiche également l'automate qui constitue l'hypothèse en cours.

En conclusion de l'exécution de l'apprentissage, Netzob affiche un automate conforme aux résultats des expériences qu'il a mené. Ainsi l'automate représenté par la figure 17 a été obtenu au terme de l'apprentissage de la grammaire du *pbot*.

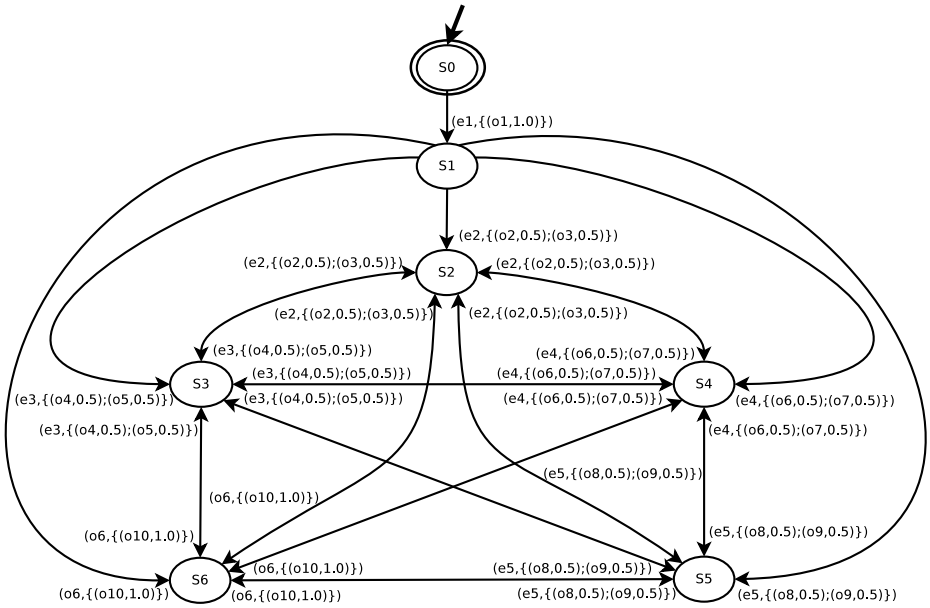

Figure 17. Grammaire inférée pour le bot pbot.

6 Conclusion

Malgré l'émergence des standards et des logiciels libres, plusieurs besoins légitimes justifient le recours à la rétro-conception de protocoles :

- dans le domaine de la sécurité, il est parfois nécessaire d'évaluer des produits dont la spécification est inconnue ;
- toujours dans ce domaine, il est souvent nécessaire d'analyser les protocoles utilisés par les logiciels malveillants pour développer des contre-mesures préventives ou de détection. Cela permet également de générer du trafic correspondant à ces menaces afin d'évaluer les produits de sécurité censés les contrées ;
- la rétro-conception peut également être utilisée pour réaliser le portage de logiciels ou de matériels sur des environnements non supportés par l'éditeur.

Ces dernières années, le domaine de l'inférence de protocoles a fait l'objet d'un nombre significatif de travaux dans le monde académique. Malheureusement, la plupart de ces travaux n'ont pas donné lieu au développement d'outils librement accessibles et utilisables « en production ». En outre, peu de travaux se sont attachés à couvrir l'ensemble des besoins (inférence du vocabulaire, de la grammaire, génération de trafic).

L'expérience montre que la plupart des experts sécurité ou des développeurs réalisant le portage de solutions propriétaires, qui sont amenés à rétro-concevoir des protocoles, n'utilisent pas ces techniques d'inférence avancées.

Nous avons, dans cet article, présenté Netzob, un outil conçu pour répondre à ce besoin. Il s'agit d'un outil libre, destiné à la rétro-conception de protocoles et à la simulation de trafic. Notre objectif est de proposer un *framework* le plus complet possible, permettant de traiter différentes sources de messages et de réaliser les différentes étapes de rétro-conception, en utilisant les techniques à la pointe de l'état de l'art. Cet outil dépasse également le simple besoin de rétro-conception : il permet la simulation de trafic suivant les protocoles inférés et, à termes, l'intégration avec les outils de *fuzzing*. Il a été développé dans le cadre de travaux de recherche menés conjointement à AMOSSYS et à Supélec et il est mis en œuvre par l'équipe technique d'AMOSSYS pour ses travaux d'audit et d'évaluation de sécurité. Il s'agit donc clairement d'un outil « opérationnel » mais qui permet également aux chercheurs de développer et de valider de nouvelles approches d'inférence protocolaire et de génération de trafic.

Netzob a atteint un niveau de maturité suffisant pour être testé par un utilisateur averti. Les perspectives d'évolutions et d'améliorations sont nombreuses :

- amélioration de l'interface graphique ;
- support d'autres sources de données (IPC Windows, API *hooking*, protocoles USB, etc.) ;
- amélioration des performances (prise en compte des architectures parallèles, des processeurs de cartes graphiques, etc.) ;
- de manière plus fondamentale, amélioration des modèles et des algorithmes utilisés (fonction de similarité sémantique ne dépendant pas seulement du découpage, grammaire plus expressive, etc.).

L'équipe travaille activement sur certains de ces sujets mais toute contribution de la communauté est la bienvenue. Nous souhaitons notamment profiter du retour d'expérience des utilisateurs afin d'améliorer l'outil et d'identifier les besoins spécifiques.

Remerciements

Les auteurs remercient toute l'équipe de Netzob et tout particulièrement Olivier Tétard, Goulven Guiheux, Alexandre Pigné et Maxime Olivier. Plus globalement, nous souhaitons remercier la société AMOSSYS

et l'équipe CIDRe de Supélec pour leur soutien et leur accompagnement dans la création du projet.

Références

1. Dana Angluin. Learning regular sets from queries and counterexamples. *Inf. Comput.*, 75 :87–106, November 1987.
2. J. Antunes, N. Neves, and P. Verissimo. Reverse engineering of protocols from network traces. In *Reverse Engineering (WCRE), 2011 18th Working Conference on*, pages 169 –178, oct. 2011.
3. Marshall A. Beddoe. Network protocol analysis using bioinformatics algorithms. In *Toorcon*, 2004.
4. Therese Berg, Bengt Jonsson, Martin Leucker, and Mayank Saksena. Insights to angluin's learning. Technical report, In International Workshop on Software Verification and Validation (SVV), 2003.
5. Therese Bohlin and Bengt Jonsson. Regular inference for communication protocol entities. Technical Report 2008-024, Uppsala University, Computer Systems, 2008.
6. Georges Bossert, Guillaume Hiet, and Thibaut Henin. Modelling to simulate botnet command and control protocols for the evaluation of network intrusion detection systems. In *Conference on Network and Information Systems Security (SAR-SSI)*, pages 1 –8, may 2011.
7. R Braden. Rfc 1122 : Requirements for internet hosts - communication layers, 1989.
8. R.G. Bucharaev. *Theorie der stochastischen Automaten*. Teubner, 1995.
9. Juan Caballero, Heng Yin, Zhenkai Liang, and Dawn Song. Polyglot : Automatic extraction of protocol format using dynamic binary analysis. In *In Proceedings of the 14th ACM Conference on Computer and and Communications Security (CCS'07)*, 2007.
10. Chia Yuan Cho, Domagoj Babi ć, Eui Chul Richard Shin, and Dawn Song. Inference and analysis of formal models of botnet command and control protocols. In *Proceedings of the 17th ACM conference on Computer and communications security*, CCS '10, pages 426–439, New York, NY, USA, 2010. ACM.
11. Paolo Milani Comparetti, Gilbert Wondracek, Christopher Kruegel, and Engin Kirda. Prospex : Protocol specification extraction. In *Proceedings of the 2009 30th IEEE Symposium on Security and Privacy*, pages 110–125, Washington, DC, USA, 2009. IEEE Computer Society.
12. Weidong Cui, Vern Paxson, Nicholas C. Weaver, and Y H. Katz. Protocol-independent adaptive replay of application dialog. In *In The 13th Annual Network and Distributed System Security Symposium (NDSS)*, 2006.
13. Drew Fisher. Reverse engineering usb devices. 28C3, http://events.ccc.de/congress/2011/Fahrplan/events/4847.en.html, 2010.
14. Andreas Hagerer, Hardi Hungar, Oliver Niese, and Bernhard Steffen. Model generation by moderated regular extrapolation. In *Proceedings of the 5th International Conference on Fundamental Approaches to Software Engineering*, FASE '02, pages 80–95, London, UK, UK, 2002. Springer-Verlag.

15. C.D. Higuera. *Grammatical inference : learning automata and grammars*. Cambridge University Press, 2010.
16. Gerard J. Holzmann. *Design and validation of computer protocols*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1991.
17. ISO. Information processing systems – OSI reference model, international standards organization. Technical Report 7498, ISO, October 1984.
18. Kevin J. Lang. Faster algorithms for finding minimal consistent dfas. Technical report, NEC Research Institute, 4 Independence Way Princeton, NJ 08540, December 1999.
19. Corrado Leita, Ken Mermoud, and Marc Dacier. Scriptgen : an automated script generation tool for honeyd. In *Proceedings of ACSAC*, 2005.
20. Zhiqiang Lin, Xuxian Jiang, Dongyan Xu, and Xiangyu Zhang. Automatic protocol format reverse engineering through context-aware monitored execution. In *In 15th symposium on Network and Distributed System Security (NDSS)*, 2008.
21. George H. Mealy. A Method for Synthesizing Sequential Circuits. *Bell System Technical Journal*, 34(5) :1045–1079, 1955.
22. Saul B. Needleman and Christian D. Wunsch. A general method applicable to the search for similarities in the amino acid sequence of two proteins. *Journal of Molecular Biology*, 48(3) :443 – 453, 1970.
23. Oliver Niese. *An Integrated Approach to Testing Complex Systems*. PhD thesis, Erlangung des Grades eines Doktors der Naturwissenschaften der Universitat Dortmund am Fachbereich Informatik, 2003.
24. Azaria Paz. *Introduction to Probabilistic Automata*. Academic Press, Orlando, FL, USA, 1971.
25. Rob Savoye. Reverse engineering of proprietary protocols, tools and techniques. FOSDEM, http://archive.fosdem.org/2009/schedule/events/499, 2009.
26. Sharon Shoham, Eran Yahav, Stephen Fink, and Marco Pistoia. Static specification mining using automata-based abstractions. In *Proceedings of the 2007 international symposium on Software testing and analysis*, ISSTA '07, pages 174–184, New York, NY, USA, 2007. ACM.
27. R. R. Sokal and C. D. Michener. A statistical method for evaluating systematic relationships. *University of Kansas Scientific Bulletin*, 28 :1409–1438, 1958.
28. Yipeng Wang, Zhibin Zhang, Danfeng Daphne Yao, Buyun Qu, and Li Guo. Inferring protocol state machine from network traces : a probabilistic approach. In *Proceedings of the 9th international conference on Applied cryptography and network security*, ACNS'11, pages 1–18, Berlin, Heidelberg, 2011. Springer-Verlag.
29. Sean Whalen, Matt Bishop, and James P. Crutchfield. Hidden markov models for automated protocol learning. In *Security and Privacy in Communication Networks*, volume 50 of *Lecture Notes of the Institute for Computer Sciences, Social Informatics and Telecommunications Engineering*. Springer Berlin Heidelberg, 2010. 10.1007/978-3-642-16161-2_24.
30. Gilbert Wondracek, Paolo Milani Comparetti, Christopher Kruegel, and Engin Kirda. Automatic network protocol analysis. In *Proceedings of the 15th Annual Network and Distributed System Security Symposium (NDSS'08)*, 2008.
31. H. Zimmermann. OSI Reference Model–The ISO Model of Architecture for Open Systems Interconnection. *IEEE Transactions on Communications*, 28(4) :425–432, 1980.