Outils pour la rétro-conception de protocoles : Analyse et classification

Julien Duchene, Colas Le Guernic, Eric Alata, Vincent Nicomette, Mohamed Kaâniche

Outils pour la rétro-conception de protocoles :
Analyse et classification

Julien Duchêne [1,2], **Colas Le Guernic** [1,3], **Eric Alata** [2], **Vincent Nicomette** [2], **Mohamed Kaâniche** [2]

1. *DGA Maîtrise de l'Information*
   Rennes, France
   prenom.nom@intradef.gouv.fr
2. *LAAS-CNRS, Univ. de Toulouse, CNRS, INSA*
   Toulouse, France
   prenom.nom@laas.fr
3. *Laboratoire Haute Sécurité, INRIA équipe TAMIS*
   Rennes, France
   colas.le-guernic@inria.fr

RÉSUMÉ. *La rétro-conception de protocoles de communication consiste à développer des méthodes et outils permettant d'inférer un modèle de ce protocole. Elle est utilisée dans de nombreux domaines d'application, allant de l'interopérabilité à l'audit de sécurité. Durant les douze dernières années, de nombreux outils ont été développés pour automatiser tout ou partie de l'inférence de protocole. Ils s'aident de différentes techniques, que les auteurs choisissent et adaptent en fonction du but final de leur rétro-conception. Le but de cet article est de présenter un état de l'art de ces différents outils, en essayant de dégager les grandes familles de techniques utilisées jusqu'à ce jour.*

ABSTRACT. *Communication protocols reverse engineering consists in developing methods and techniques enabling the inference of a model of these protocols. It is used for different purposes such as to ensure interoperability or to perform security audits. During the last twelve years, numerous tools have been developed to automate, totally or partially, the inference of protocols. The tools rely of various techniques, that are selected and tuned depending on the goal of the reverse engineering activity. The aim of this paper is to present the state of the art of reverse engineering tools, and to identify the major categories of techniques used by these tools.*

MOTS-CLÉS : *rétro-conception, inférence de protocole, analyse de traces réseau, analyse d'applications binaires.*

KEYWORDS: *reverse engineering, protocol inference, network trace analysis, binary application analysis.*

1. Introduction

Les protocoles de communication permettent à plusieurs composants d'échanger des messages de manière cohérente. Ils sont largement utilisés en informatique, dans les réseaux et en télécommunication. Un protocole peut faire l'objet d'un standard ouvert ou être tout simplement masqué aux yeux de l'utilisateur du composant. La première approche peut être choisie pour aider à la création de composants capables d'établir une communication cohérente avec d'autres composants qui supportent déjà le protocole. Quant à la seconde approche, elle peut être choisie par exemple quand le développeur a construit son propre protocole et ne juge pas nécessaire d'en dévoiler les détails, ou bien quand le protocole porte sur une partie sensible de la communication entre les composants, le développeur ne souhaitant pas divulguer ces détails. La rétro-conception rentre principalement dans le cadre de cette seconde approche.

La rétro-conception de protocoles consiste en l'obtention, par un analyste, d'un modèle des communications établies entre plusieurs composants sans disposer au préalable, de tous les détails concernant le protocole.

Un des projets majeurs de rétro-conception de protocoles est le projet Samba (Samba Team, 2017) qui offre une implémentation open-source des protocoles *SMB* et *CIFS* pour des clients Linux [1] permettant l'interopérabilité entre les systèmes Windows et Linux. À ses débuts en 1992, ce projet se basait principalement sur une rétro-conception manuelle du protocole, un travail complexe, fastidieux et chronophage, dont les résultats dépendent fortement du niveau de compétence des analystes. De plus, pour rester pertinent le projet a dû suivre les nombreuses évolutions du protocole. Toutefois, l'**interopérabilité** n'est qu'un des domaines concernés par la rétro-conception de protocoles.

La **simulation de protocoles réseau** (Leita *et al.*, 2005 ; Cui *et al.*, 2006 ; Newsome *et al.*, 2006 ; Caballero, Song, 2007) est également un domaine pour lequel la rétro-conception peut être utile. Elle permet dans ce cas de créer des simulateurs de services réseau. Les intérêts de l'application de la rétro-conception à cette fin sont nombreux. Par exemple, un simulateur permet de réaliser rapidement des tests alors que communiquer directement avec le véritable composant peut être lourd voire impossible dans certains cas. Il permet également d'analyser le comportement de tous les composants sur un réseau en menant des études statistiques. Pour finir, il permet de rejouer, dans des environnement variés, des traces réseau en adaptant le contenu des messages. Le rejeu est utile, par exemple, pour analyser une attaque réseau dans un environnement contrôlé. Plus concrètement, la rétro-conception pour obtenir un simulateur peut être utile dans le cas de pots de miels qui vont ainsi pouvoir entrer en interaction avec des attaquants et enregistrer leur comportement.

1. Jusqu'à ce que Microsoft ouvre sa spécification en 2006. https://msdn.microsoft.com/en-us/openspecifications/default

La rétro-conception de protocoles peut également être utilisée pour l'**audit de sécurité de logiciels** (Guihery, Bossert, 2012b ; Caballero *et al*., 2007 ; Cui *et al*., 2008). Cette application est très proche de la précédente mais se distingue par son objectif qui est de solliciter un composant avec une grande variété de communications différentes pour vérifier que le composant se comporte normalement quelque soit le contexte de la communication. Ainsi, le modèle du protocole peut être utilisé pour fabriquer des fuzzers intelligents qui permettent de tester la robustesse de l'implémentation du protocole. Ces fuzzers génèrent des messages à destination du composant en modifiant certains champs des messages. Les messages qui déclenchent une vulnérabilité peuvent être traduits en signatures et ajoutés à un système de détection d'intrusion, par exemple.

L'**analyse de protocoles de malware** (Bossert *et al*., 2011 ; Caballero *et al*., 2009 ; Caballero Bayerri, 2010) s'appuie sur la rétro-conception. En effet, bon nombre de malware, comme les bots, utilisent des protocoles pour communiquer avec leur serveur de *Command & Control*. La rétro-conception de ces protocoles peut être particulièrement utile dans la mesure où elle peut permettre d'identifier des informations cruciales concernant la localisation du maître du botnet, une date, un type d'attaque imminente, ses cibles, etc. Elle permet d'anticiper sur l'occurrence proche d'attaques et ainsi d'améliorer la réponse à incident.

Enfin, la rétro-conception peut être utile pour le **test de conformité de protocoles réseau**. Il s'agit de vérifier qu'un logiciel implémente correctement un protocole réseau dont les spécifications sont connues. La rétro-conception permet d'obtenir un modèle du protocole, tel qu'il est utilisé par le logiciel à l'étude, et de vérifier que ce modèle est en accord avec la spécification.

Un état de l'art de la rétro-conception de protocoles (Li, Chen, 2011) a été présenté par Li et Chen en 2011. Cependant, cet article ne prend pas en compte un certain nombre de travaux intéressants. Les informations fournies sont succinctes et ne permettent pas d'avoir une analyse synthétique et comparative des différentes approches utilisées pour l'inférence de protocole de communication. Par ailleurs, de nouvelles contributions sont apparues depuis 2011. L'état de l'art réalisé par Narayan *et al*. (Narayan *et al*., 2015) en 2015 est beaucoup plus complet. Tout comme Li et Chen et Narayan *et al*., nous nous intéressons aux contributions qui ont donné lieu à des outils. Nous nous intéressons de plus aux outils d'inférence de structures de données qui pour certains sont directement issus d'outils d'inférence de format de message. Par ailleurs, nous présentons également les diverses étapes nécessaires à l'inférence de protocole ainsi que des défis clés qui y sont associés. Concernant la classification en terme de techniques utilisées pour l'inférence du format des messages et/ou de la grammaire de protocole, nous distinguons, comme Narayan *et al*., celles basées sur l'observation de trafic sur le réseau (appelée inférence réseau), de celles basées sur l'analyse de traces d'exécution (appelée inférence applicative). Notre classification considère en outre un second critère de distinction indépendant du précédent : d'une part, les approches nécessitant, et tirant profit, de nouvelles observations pendant l'inférence (inférence active), et, d'autre part, les approches exploitant uniquement un

corpus d'observations initiales (inférence passive). Ainsi, le but de ce article est de faire un état de l'art de l'évolution des différentes contributions du domaine de la rétro-conception de protocoles. La section 2 présente la terminologie associée à la rétro-conception de protocoles. La section 3 est dédiée aux différentes étapes de la rétro-conception de protocoles et leurs défis associés. L'évolution et l'analyse comparative des outils de rétro-conception de protocoles et des techniques associées sont détaillées dans la section 4. Enfin, la section 5 présente quelques conclusions et perspectives.

2. Définitions

De nombreuses études (Beddoe, 2004b ; Cui *et al*., 2006 ; Newsome *et al*., 2006 ; Caballero *et al*., 2007 ; Cui, Kannan, Wang, 2007 ; Bohlin, Jonsson, 2008) débutent par une définition des termes importants en rétro-conception. Ces termes visent à définir à la fois les composants qui entrent en jeu dans la communication ainsi que la structure de la communication et des messages échangés. À notre connaissance, il n'existe pas de consensus concernant ces définitions. Dans cet article, nous utilisons les termes les plus couramment utilisés dans la littérature.

Un *protocole* est composé de *messages*, correspondant à l'unité de données élémentaires du protocole, aussi appelé *PDU* (*Protocol Data Unit*). Les messages sont regroupés en *classes de messages* identifiés par un *type*. Un message est composé de *champs*, encodés selon un certain *format*. Les messages sont échangés selon une *grammaire*. Notons que, dans la suite, nous considérerons qu'une structure de données telle qu'utilisée dans une application correspond simplement à un format de message.

Ces termes peuvent être illustrés avec le sous-ensemble du protocole TCP qui correspond à l'établissement d'une connexion, associé aux classes de messages suivantes : {`Syn, Syn-Ack, Ack`}. La grammaire du protocole spécifie que, lorsque le composant A tente d'établir une connexion avec le composant B : 1) A doit envoyer avant tout le message `Syn`, 2) B doit répondre avec le message `Syn-Ack` et 3) A doit finaliser l'établissement de la connexion avec le message `Ack`. Le format du message de classe `Syn` est fourni à la figure 1. Par exemple, le champs `flags` commence à l'offset 0x35 et possède une taille de 3 octets. Notons que cette classe de messages correspond effectivement à un ensemble de messages. La caractérisation du message en classe `Syn` est due au fait que le bit *Syn* du champ `flags` est à 1 et que les autres bits sont à 0, soit la valeur 0x2 pour le champ `flags`. Si ce champ doit avoir une valeur précise, tous les autres champs sont instanciés au moment de l'envoi, leur valeur pouvant changer d'une communication à une autre.

Les techniques de rétro-conception de protocoles sont habituellement classées en deux catégories : d'une part, la rétro-conception basée sur une analyse des messages échangés entre deux applications, et, d'autre part, celle basée sur une analyse d'une application utilisant ce protocole. Les messages étant généralement échangés sur le réseau, une séquence de messages échangés entre deux application sera appelée une *trace réseau*. L'analyse d'une application consiste pour les outils étudiés soit en l'ana-

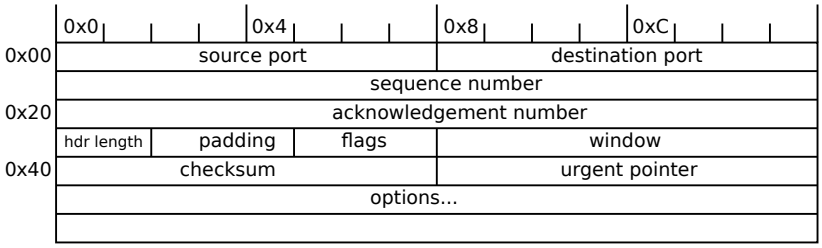

Figure 1. Format d'un paquet `Syn` *du protocole TCP, où la structure des champs est appliqué sur leur représentation en mémoire à l'adresse 0x00*

lyse de son code (source ou binaire) soit en l'analyse d'une *trace d'exécution* (séquence des instructions assembleur exécutées par l'application durant le traitement d'éléments du protocole à inférer). Par souci de concision, dans la suite de ce papier, l'inférence basée sur l'analyse d'une trace réseau sera appelée *inférence réseau*, l'inférence basée sur l'analyse d'une application utilisant ce protocole sera appelée *inférence applicative*. Il est également possible de distinguer ces techniques selon l'approche (active ou passive) utilisée pour l'inférence. L'*inférence active* stimule le système pour découvrir de nouvelles informations ou valider des hypothèses. A contrario, l'*inférence passive* est réalisée à partir d'observations capturées initialement, et donc sans nouvelle stimulation pendant la phase d'inférence. Ainsi, un outil peut pratiquer une *inférence réseau active*, une *inférence réseau passive*, une *inférence applicative active* ou une *inférence applicative passive*. Cependant, nous constatons que la majorité des outils étudiés utilisent une approche passive.

En fonction du but poursuivi par l'inférence de protocole, un analyste n'aura pas nécessairement besoin de tous les détails du fonctionnement du protocole. Ainsi, des outils laissent de côté certains aspects de l'inférence de protocole. Dans la prochaine section, nous détaillons les challenges soulevés lors des différentes étapes de la rétro-conception de protocoles.

3. Challenges associés aux étapes de la rétro-conception de protocoles

Les défis imposés par la rétro-conception de protocoles sont nombreux et variés. Ils ne sont pas tous traités systématiquement par chaque étude et il n'existe pas de solution ultime capable de traiter à la fois tous ces défis. Cette section est destinée à énumérer ces défis, identifiés en parcourant la littérature ou identifiés par notre propre expérience dans ce domaine. Pour débuter cette section, nous présentons les étapes classiques suivies lors de la rétro-conception de protocoles. Ensuite, pour chacune de ces étapes, nous identifions un défi principal qui influence la qualité globale de l'inférence.

La rétro-conception ne débute pas directement par les activités de reconstruction du protocole. Ces activités sont précédées d'une identification et d'une caractérisation de l'environnement. Le bon déroulement de cette préparation dépend grandement

du niveau de l'analyste, comme pour les étapes de la rétro-conception et elle est primordiale pour aboutir à une rétro-conception efficace, rapide et complète. Cependant, ces aspects ne seront pas développés dans cet article et nous nous focaliserons sur les étapes de la rétro-conception elle-même. Sur la base de la connaissance de l'environnement, l'analyste peut débuter cette rétro-conception, en commençant par une étape d'observation qui consiste à mettre en place l'outillage et récupérer les données qui permettront de réaliser la rétro-conception du protocole. La seconde étape vise à pré-traiter les observations pour les associer aux différents messages du protocole. La dernière étape permet d'inférer le format des messages ou la grammaire du protocole directement à partir de ces données. Bien entendu, ces étapes ne sont pas exécutées en boucle ouverte. Autrement dit, un résultat d'une de ces étapes peut amener l'analyste à revoir sa stratégie concernant les premières étapes. De plus, l'expérience et l'intuition de l'analyste peuvent réduire l'effort dédié à certaines de ces étapes. Par exemple, la rétro-conception d'un vers qui utilise un canal *Command & Control* sur `IRC` sera réalisé rapidement par un analyste qui connaît déjà ce type de protocoles.

La figure 2 illustre les étapes de la rétro-conception en mettant en évidence les sous-étapes pour lesquelles nous avons identifié des défis majeurs. La suite est consacrée à la présentation de ces étapes.

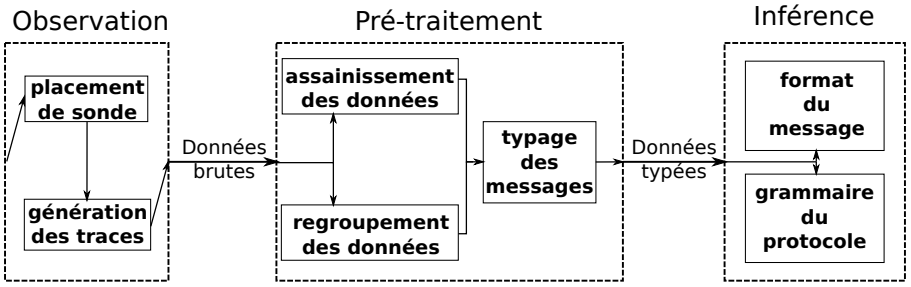

Figure 2. Étapes de rétro-conception de protocoles avec les défis associés

3.1. Étape d'observation

L'inférence, telle qu'abordée dans cet article, se base sur l'obtention d'un ensemble de traces par observation du système. L'obtention de ces traces amène deux défis importants. Le premier concerne le **placement de la sonde** et le second concerne la **génération de ces traces**.

Le placement de la sonde est déterminant pour l'obtention des données indispensables à la rétro-conception des protocoles. Les défis ne sont pas les mêmes suivant l'approche réseau ou applicative de l'inférence.

Pour l'inférence réseau, le placement de la sonde peut sembler évident mais il peut aussi devenir délicat lorsque le protocole utilise plusieurs canaux de communication. Dans ce cas, il faut identifier le canal qui va véhiculer les messages liés au

protocole à rétro-concevoir et il peut même être nécessaire d'envisager l'usage de plusieurs sondes, ce qui nécessite la synchronisation des différentes sondes pour conserver l'ordre des messages. Le placement doit également permettre de collecter tous les messages nécessaires à la reconstruction de tout le protocole ou de la partie intéressante. Une faiblesse au niveau de ce placement peut se traduire par des messages non observés, la reconstruction du protocole sera donc partielle. Si dans certains domaines d'application cette faiblesse n'est pas forcément gênante, elle peut empêcher son application dans d'autres domaines où le rejeu, par exemple, est un objectif. De plus, la difficulté du placement est accentuée par l'usage de mécanismes cryptographiques qui rendent l'interprétation des traces difficile voire impossible. Il est alors nécessaire soit de casser le mécanisme cryptographique, soit de se placer aux extrémités du canal de communication, c'est-à-dire de créer une sonde placée dans l'application aux interfaces des librairies cryptographiques pour retrouver les messages en clair. Ainsi, outre le placement de la sonde, une difficulté technique s'ajoute parfois à ce défi et concerne le développement de la sonde pour l'observation.

Cette difficulté est plus poussée encore lors de l'inférence applicative qui s'appuie sur des traces d'exécution de l'application. Ainsi, la sonde doit produire la séquence des instructions exécutées par l'application lors du traitement de messages, mais aussi l'état de la mémoire du système. Plusieurs solutions existent (débogage, instrumentation, virtualisation...) mais elles doivent toutes faire face à différents défis tels que : la présence de protections contre l'analyse (contrôle d'intégrité du code, antidebogue...), ou tout simplement le ralentissement de l'application qui peut provoquer des pertes de messages suite à un dépassement de délai.

La trace (réseau ou d'exécution) constitue la donnée principale pour rétro-concevoir le format du message ou la grammaire du protocole. La qualité de cette trace dépend du positionnement de la sonde, comme abordé dans le paragraphe précédent. Par contre, cette qualité peut également être influencée par la durée de l'observation. Évidemment, pour un même protocole, une observation courte fournira probablement une trace plus courte qu'une observation longue. Cela ne veut pas pour autant dire qu'une observation courte est insuffisante. Effectivement, il suffit simplement qu'elle ait capturé l'ensemble des états possibles de la communication et les liens entre ces états, de sorte à inclure tous les formats possibles de messages et tous les scénarios d'échange des messages possibles. Une trace trop pauvre amène à ne reconstruire qu'une partie de la spécification du format du message ou de la grammaire du protocole. La question est donc de savoir à partir de quel moment la trace est assez riche et diversifiée. Bien entendu, cela dépend de la nature des composants qui communiquent et de la fréquence des communications. Dans certains cas, l'analyste peut avoir une approche active : soit en forçant les échanges, soit en perturbant l'environnement (ce qui lui donne des indications sur la nature des échanges), soit en employant son propre composant pour communiquer.

3.2. Étape de pré-traitement

La phase de pré-traitement dépend largement de l'étape précédente. En particulier, pour l'inférence réseau comme applicative, l'analyste n'a pas forcément pu placer la sonde sur l'emplacement idéal pour l'observation et il a peut être dû cibler un protocole sous-jacent. Du coup, les observations peuvent contenir des informations inintéressantes pour l'analyste et les messages pertinents pour la rétro-conception peuvent être : 1) encapsulés dans un autre protocole mais aussi 2) découpés en plusieurs blocs transférés en plusieurs fois ou 3) noyés dans une observation de multiples protocoles.

Un premier défi dans cette étape est donc de correctement nettoyer les observations pour pouvoir ensuite reconstruire les messages, cela correspond à l'**assainissement des données**. Cela implique d'être capable de discerner les informations pertinentes des informations inintéressantes. Ce défi concerne à la fois l'inférence réseau et applicative. Toutefois, l'inférence applicative doit faire face à un défi supplémentaire qui concerne l'identification dans la trace d'exécution des structures de contrôle (sauts, boucles, tests...), qui reflètent fortement la structure encapsulée et itérative de certains messages.

Dans le cas où les messages sont transférés en plusieurs blocs, un second défi consiste donc à regrouper les traces permettant de reformer les messages pertinents pour l'analyse. Pour l'inférence réseau, par exemple, si l'observation se fait à la couche TCP, alors les paquets de segments TCP doivent être regroupés pour reconstituer des messages d'un protocole de niveau applicatif (couche OSI 7). Pour l'inférence applicative, si la trace d'exécution du traitement complet d'un message est découpée en plusieurs traces d'exécution, alors ces traces doivent être unifiées. Par ailleurs, les traces (réseau ou d'exécution) peuvent contenir des données concernant plusieurs messages, il faut donc découper les données par message. Ces deux éléments constituent le **regroupement des données**.

Que ce soit pour l'inférence du format des messages ou de la grammaire du protocole, il est nécessaire de regrouper les messages en classes de messages. Ce regroupement est nécessaire pour comparer des messages ayant la même sémantique. Ce point revient à trouver une fonction permettant, à partir d'un message représenté par une séquence d'octets, de trouver le type du message, sachant que cette fonction de **typage** doit prendre en compte le degré de parenté entre les messages.

3.3. Étape d'inférence

L'**inférence du format des messages** vise, à partir des messages d'un même type, à identifier leur structure et l'**inférence de la grammaire du protocole** vise, à partir de séquences de types de messages, à aboutir aux règles liées aux échanges de ces classes de messages. La réalisation de cette étape fait face à deux défis importants.

Que ce soit pour l'inférence du format des messages ou de la grammaire du protocole, il est important d'être capable d'identifier des dépendances, entre les champs d'un message ou entre les messages eux-mêmes. Sinon, le résultat de la rétro-concep-

tion risque de contenir des messages ou échanges de messages qui ne respectent pas ces dépendances.

Le résultat final de la rétro-conception est d'obtenir une spécification du format des messages et de la grammaire. Cette spécification est représentée avec un modèle. Il est donc nécessaire d'utiliser un modèle suffisamment expressif pour s'approcher au mieux de la spécification originale. Par exemple, la plupart des formats de message ou grammaires de protocoles complexes possèdent une structure arborescente ou récursive. La présence de cette structure implique qu'un message (dans le cas de l'inférence de grammaire) ou que la valeur d'un champ (dans le cas de l'inférence du format des messages) dépend des messages ou champs précédents. Ce type de dépendances est très difficile à exprimer avec des automates à états finis par exemple. Le choix du bon modèle est donc un défi majeur. Concrètement, cela revient à identifier l'outil adéquat pour l'inférence. Un mauvais choix risque d'aboutir à une spécification adaptée uniquement à représenter les observations sans capacité à généraliser. De plus, les observations sont généralement incomplètes, ainsi, les outils doivent utiliser des approches suffisamment robustes pour généraliser les modèles obtenus avec un minimum de sur-approximation.

Dans la littérature, nous avons identifié trois classes de modèles utilisés :

– **Structures énumérées** : les modèles associés sont les templates qui sont généralement utilisés pour représenter le format des messages. Ils sont par ailleurs annotés dans une étape de post-traitement pour pouvoir représenter des dépendances entre champs.

– **Structures itératives** : les modèles associés sont les langages réguliers qui sont utilisés à la fois pour le format des messages et pour la grammaire de protocole. Ils sont souvent représentés sous la forme d'automates finis déterministes ou de machines de Mealy pour la grammaire de protocole, et sous forme d'arbres pour le format des messages, où chaque nœud correspond à un champ du message. Ces arbres sont aussi annotés pour garder des dépendances entre champs. Contrairement aux templates, cette représentation permet de mettre en évidence l'encapsulation de plusieurs couches protocolaires dans le message.

– **Structures récursives** qui ne sont utilisées que pour la grammaire de protocole. Elles peuvent être représentées sous la forme d'automates à pile ou de chaînes de Markov cachées, seule cette dernière représentation est effectivement utilisée par l'outil *PRISMA* (Krueger *et al.*, 2012).

Si ces modèles sont issus de l'ingénierie des protocoles, ils sont adaptés (par des étapes de post-traitement) à la rétro-conception des protocoles, et au but final de cette rétro-conception (cf. section 1). De plus, certains modèles souffrent de restrictions liées à leur méthode d'apprentissage. Ainsi, le format BNF (Backus-Naur Form) (équivalent à des automates à pile) est utilisé par l'outil *AutoFormat* (Lin *et al.*, 2008) mais sa méthode d'apprentissage ne lui permet d'inférer que des structures itératives.

Rappelons que la synthèse de cette section est représentée dans le schéma de la figure 2 qui reprend les trois étapes présentées ainsi que la plupart des défis associés.

4. Revue des outils existants et classification

De nombreuses contributions ont fait évoluer la rétro-conception de protocoles. Les outils présentés dans cet article se basent principalement sur deux contributions majeures. Le premier outil, *PI Project* (Beddoe, 2004b), a introduit les algorithmes de bio-informatique pour l'inférence réseau de format de messages. Le second outil, *Polyglot* (Caballero *et al*., 2007), a présenté une nouvelle technique d'inférence de format de message à partir de traces d'exécution d'une application traitant une entrée. Les tableaux 1 et 2 présentent les différentes publications associées aux outils d'inférence réseau et applicative respectivement. Les outils en gras sont ceux utilisant une approche active. Ces diagrammes montrent que ces outils se sont majoritairement concentrés sur l'inférence de format de messages, l'argument principal développé dans la littérature est qu'avant de pouvoir inférer la grammaire du protocole, il faut que le format des messages soit connu. De plus, nous constatons que l'inférence active commence tout juste à voir le jour dans ce domaine.

4.1. Inférence réseau

Dans (Beddoe, 2004a ; 2004b), l'auteur présente l'outil, ***PI Project***[2] pour réaliser la rétro-conception du format de messages à partir de traces réseau. Cet outil laisse donc de côté l'inférence de la grammaire du protocole. Il emploie l'algorithme *Needleman & Wunsch* (Needleman, Wunsch, 1970) pour aligner les séquences d'octets entre les messages pris deux à deux. L'alignement optimal permet d'identifier les parties communes entre deux messages qui correspondent, selon l'outil, aux mêmes champs. Ce résultat est utilisé pour construire un arbre correspondant à la classification hiérarchique des messages en fonction de leur degré de similitude (les messages les plus similaires se trouvent sur des feuilles voisines de l'arbre). L'algorithme employé est UPGMA (Nei *et al*., 1983). Par contre, l'utilisateur de cet outil doit partitionner manuellement cet arbre pour identifier les classes des messages. La majorité des outils d'inférence réseau de format de messages, parus par la suite, s'appuie sur ces travaux.

Peu après, Leita *et al.* ont proposé l'outil ***ScriptGen*** (Leita *et al*., 2005 ; Leita, 2008), qui a pour objectif de générer des scripts de simulation réseau du comportement d'un serveur, pour le pot de miel *Honeyd*. Contrairement à *PI Project*, l'inférence de la grammaire du protocole est nécessaire à cet outil pour atteindre cet objectif. Tout d'abord, *ScriptGen* opère une phase de pré-traitement correspondant d'une part au filtrage des paquets réseau pour conserver uniquement les paquets TCP des communications à simuler et d'autre part à la reconstruction des messages à partir de ces paquets. Concernant l'inférence du format des messages, l'approche employée est similaire à celle utilisée dans *PI Project*. Par contre, le partitionnement en classes de messages est réalisé automatiquement en s'appuyant sur une valeur seuil. Concernant l'inférence de la grammaire, *ScriptGen* utilise une approche basée sur l'inférence

2. Disponible en source ouverte à http://www.4tphi.net/~awalters/PI/PI.html

Tableau 1. Évolution des contributions pour l'inférence de protocole basée sur l'analyse du réseau. Les outils utilisant une approche active sont mis en gras

	Inférence réseau	
	Format du message	Grammaire du protocole
2004	PI Project (Beddoe, 2004b ; 2004a)	
2005	ScriptGen (Leita *et al*., 2005)	ScriptGen (Leita *et al*., 2005)
2006	**RolePlayer (Cui *et al*., 2006)**	
2007	Discoverer (Cui, Kannan, Wang, 2007)	
2008		
2009		
2010	ASAP (Krueger *et al*., 2010)	Veritas (Y. Wang *et al*., 2010)
2011	ReverX (Antunes *et al*., 2011) Netzob (Bossert *et al*., 2011)	ReverX (Antunes *et al*., 2011) Veritas (Y. Wang *et al*., 2011)
2012	Netzob (Guihery, Bossert, 2012b ; 2012a)	**Netzob (Guihery, Bossert, 2012b ; 2012a)** PRISMA (Krueger *et al*., 2012)
2013		
2014	**Netzob (Bossert *et al*., 2014)**	**Netzob (Bossert *et al*., 2014)**

Tableau 2. Évolution des contributions pour l'inférence de protocole basée sur l'analyse d'application. Les outils utilisant une approche active sont mis en gras

	Inférence applicative	
	Format du message	Grammaire du protocole
2005		
2006	FFE/x86 (Lim *et al*., 2006) **Replayer (Newsome *et al*., 2006)**	
2007	Polyglot (Caballero *et al*., 2007) Rosetta (Caballero, Song, 2007)	
2008	AutoFormat (Lin *et al*., 2008) Tupni (Cui *et al*., 2008) Prospex (Wondracek *et al*., 2008) **ConfigRE (R. Wang *et al*., 2008)**	
2009	ReFormat (Z. Wang *et al*., 2009) Dispatcher (Caballero *et al*., 2009)	Prospex (Comparetti *et al*., 2009)
2010	REWARDS (Lin *et al*., 2010)	**MACE (Cho *et al*., 2010)**
2011	Howard (Slowinska *et al*., 2011)	**MACE (Cho *et al*., 2011)**
2012	ARTISTE (Caballero *et al*., 2012)	
2013	Dispatcher (Caballero, Song, 2013)	
2014		
2015	ARGOS (Zeng, Lin, 2015)	

de langages réguliers. Le modèle résultant est un automate déterministe qui contient uniquement les réponses les plus fréquemment retournées par le serveur.

En parallèle à ces travaux, Cui *et al.* ont présenté leur outil ***Roleplayer*** (Cui *et al.*, 2006) qui a pour but de rejouer des communications dans un nouvel environnement, notamment pour étudier plus en détail le déroulement d'une attaque réseau. Le rejeu nécessite d'adapter une trace aux caractéristiques de l'environnement (par exemple, changement d'adresses IP, de numéro de séquencement, etc.). Il s'agit donc d'avoir une connaissance précise des champs des messages et de leurs dépendances. Pour traiter ce défi, *Roleplayer* utilise une inférence active durant laquelle des communications sont initiées par l'outil lui-même pour identifier ces dépendances. En appliquant également un algorithme d'alignement, il est capable d'apprendre des dépendances entre les champs d'un message (par exemple, la taille, les valeurs cookies, etc.) sans pour autant être capable d'identifier les encapsulations de champs. S'appuyer sur les résultats de cet apprentissage permet d'analyser plus finement les attaques qui ne contiennent pas beaucoup d'échanges.

Face aux difficultés rencontrées pour identifier correctement les champs d'un message, Cui *et al.* ont proposé une nouvelle approche dans l'outil ***Discoverer*** (Cui, Kannan, Wang, 2007). Leur inférence ne repose plus sur l'algorithme de bio-informatique *Needleman & Wunsch*. Ils supposent, à présent, connus les délimiteurs des champs (espaces, retours à la ligne, indentations, virgules, etc.), ce qui simplifie grandement le problème. Suite à ce découpage, l'outil enchaîne avec une classification hiérarchique et récursive des messages qui présente l'avantage de reconnaître des situations d'encapsulation des champs. Pour finir, une série d'heuristiques est appliquée pour retrouver des dépendances entre les champs.

Alors que la recherche sur l'inférence de protocole se focalisait sur l'inférence du format des messages, l'outil ***Veritas*** (Y. Wang *et al.*, 2010 ; 2011) de Wang *et al.* vise principalement à construire la grammaire du protocole. Sans nécessiter le format des messages, il lui est indispensable de connaître la classe de chaque message. Pour identifier ces classes, l'outil se base sur les séquences d'octets les plus fréquemment observées dans leur en-tête (les n premiers octets). Les auteurs font ainsi l'hypothèse que l'en-tête d'un champ est localisée au début du message et que celui-ci contient des séquences d'octets caractéristiques de la classe du message. Non seulement cette hypothèse est généralement vérifiée pour faciliter l'interprétation du message, mais de plus, en procédant de cette manière, il réduit considérablement le volume des données à traiter. Les séquences d'octets fréquentes servent donc à classer les messages. Il enchaîne ensuite avec l'inférence de la grammaire du protocole. Pour cette étape, les auteurs supposent que les messages qui peuvent être émis dépendent uniquement du dernier message observé et non de tout l'historique des échanges. Le modèle construit par cet outil est donc naturellement un automate étiqueté avec les classes de messages. Notons qu'avec cette approche, *Veritas* exclut volontairement les séquences d'octets rares qui peuvent correspondre à des données ou à un comportement inhabituels du serveur. Dans ce dernier cas, la grammaire du protocole obtenue est partielle et ne représente pas tous les comportements inclus dans les échanges observés.

Pour leur part, Krueger *et al.* ont développé l'outil ***ASAP*** (Krueger *et al.*, 2010) pour aider l'analyse de données issues de pots de miel, la conception d'IDS (systèmes de détection d'intrusions) et l'analyse de malware. Dans ce contexte, cet outil se focalise plus sur la classification des messages qu'à l'obtention du format exact de ces messages. Il utilise aussi un découpage des messages en champs, en fonction de délimiteurs prédéfinis pour des messages au format texte ou en séquences d'octets de taille fixe pour les messages au format binaire. Chaque message peut alors être caractérisé en fonction des champs qu'il contient et représenté sous la forme d'un vecteur. Les classes de messages sont identifiées comme étant des messages types qui constituent une base pour la génération de tous les messages observés, au sens algébrique. Au delà des choix concernant les outils statistiques et la manière de découper les messages, *ASAP* se distingue donc principalement des autres outils par l'usage d'une approche algébrique. Par contre, le format obtenu n'est pas très précis et ne met pas en évidence les liens entre les champs, leur ordre d'apparition ou les champs qui seraient à cheval sur deux lignes ou deux séquences d'octets.

Par la suite, les développeurs de *ASAP* ont étendu leur outil en y ajoutant une capacité d'inférence de la grammaire de protocole, ce qui a donné lieu à la création de l'outil ***PRISMA*** [3] (Krueger *et al.*, 2012). Cet outil emploie globalement la même stratégie que *ASAP*. Par contre, l'analyse algébrique des messages est remplacée par une classification basée sur une distance pour regrouper les messages en classes. Ensuite, l'outil analyse les séquences de classes de messages pour en déduire une chaîne de Markov cachée. L'usage de ce modèle présente l'avantage de pouvoir représenter des changements d'état du protocole qui ne se traduisent pas nécessairement par l'envoi d'un message.

À l'opposé des approches précédentes, l'outil ***ReverX*** [4] (Antunes *et al.*, 2011) utilise une approche unique : il traite le format des messages de la même manière que la grammaire du protocole. À l'instar de *Discoverer*, cet outil se base sur une liste connue de délimiteurs pour découper les messages en champs. Ces séquences sont assemblées sous la forme d'un arbre qui est à son tour replié sur lui-même. Ce repli aboutit à un automate contenant des boucles qui sont utiles pour généraliser le modèle à des messages non observés. Cet automate correspond ainsi aux formats des messages. La grammaire du protocole est reconstruite de la même manière, en partant des séquences de classes de messages. Cet outil offre des résultats intéressants, notamment au niveau de la généralisation du format des messages, qui constitue un défi important. Par contre, la stratégie qu'il emploie pour découper les messages en champs ne le rend pas adapté à l'analyse de messages binaires.

Pour terminer, citons l'outil ***Netzob*** [5] qui a fait l'objet de nombreux travaux différents et qui a connu de multiples évolutions ces dernières années. Il a dans un premier temps été conçu pour la modélisation de botnets (Bossert *et al.*, 2011). Les auteurs

3. Disponible en source ouverte à https://github.com/tammok/PRISMA
4. Disponible en source ouverte àhttps://github.com/jasantunes/reverx
5. Disponible en source ouverte à https://www.netzob.org/

utilisent une variante d'une machine de Mealy pour représenter la grammaire du protocole, en enrichissant cette machine par des informations sur le temps d'attente entre deux messages observés. Cette grammaire de protocole est apprise de manière active à partir de l'algorithme L^*. Cet automate est ensuite rendu non-déterministe en incorporant des probabilités d'émission de messages étant donné un état et un message reçu. Les techniques utilisées pour le regroupement des messages et l'inférence de leur format sont abordés dans (Guihery, Bossert, 2012b ; 2012a). Ainsi, l'algorithme *Needleman & Wunsch* est utilisé pour effectuer l'inférence de format de messages et la classification des messages à l'instar de l'outil *PI Project* (Beddoe, 2004b). Les auteurs introduisent de plus une heuristique nommée *orphan reduction* qui leur permet d'effectuer un alignement local entre deux messages, sans considérer l'intégralité du contenu de ces messages. Ensuite, une étape d'inférence de la sémantique des champs est réalisée, à partir d'heuristiques similaires à celles présentes dans *Discoverer* (Cui, Kannan, Wang, 2007). Ils conservent cependant dans leur format, les différents attributs inférés, tels que l'encodage, car ces informations peuvent aider un analyste à peaufiner l'inférence. Alors que l'inférence de la grammaire du protocole est basée sur une approche active, celle du format des messages est entièrement passive. Dans l'article (Bossert *et al*., 2014), ils apportent des améliorations majeures à leurs résultats en utilisant une approche active. Ils insèrent des données connues dans l'application implémentant le protocole puis recherchent ces mêmes données dans les différents messages. Ils utilisent aussi une corrélation temporelle afin de mieux associer les messages avec des actions sur l'application. Ces deux éléments leur permettent d'une part de délimiter de façon certaine des champs dynamiques, et d'autre part d'associer une sémantique aux classes de messages. Cependant, cette approche nécessite un travail supplémentaire qui peut être complexe à automatiser de la part de l'analyste. Enfin, cet article présente également une étude comparative des résultats avec ceux de *PRISMA* (Krueger *et al*., 2012) et de *ScriptGen* (Leita *et al*., 2005), chose qui n'avait pas été réalisée auparavant. Ces travaux ont été complétés dans la thèse de G. Bossert (Bossert, 2014), notamment en ce qui concerne la formalisation des modèles utilisés pour le format des messages et la grammaire du protocole, ainsi que le développement de techniques actives pour l'inférence de la grammaire. Cette nouvelle inférence leur permet de découper l'apprentissage de l'automate en sous automates liés à des actions sur le système. Cela lui permet de réduire drastiquement le temps d'apprentissage de la machine de Mealy. Il est à noter que l'outil *Netzob* possède un greffon d'exportation des résultats sous différents formats ainsi qu'un greffon de *fuzzing* et de simulation réseau intégré. De plus, c'est à notre connaissance, le seul outil utilisé en production et encore maintenu à ce jour. Il fait parti avec *PI Project*, *ReverX* et *ASAP/PRISMA* des rares outils disponibles en source ouverte.

Synthèse

Le tableau 3 résume les différentes approches utilisées par les outils pour rétroconcevoir des protocoles par l'analyse de traces réseau en précisant pour chaque outil si l'approche est active ou passive, le type de mesure utilisée pour regrouper les messages, le modèle utilisé pour le format des messages et sa capacité à identifier des

dépendances entre champs, ainsi que le modèle utilisé pour l'inférence de grammaire de protocole. L'inférence réseau se base principalement sur : 1) des techniques d'alignement de séquences pour le découpage des messages en champs et leur regroupement en classes de messages, 2) des heuristiques pour l'inférence de la sémantique des champs et des dépendances intra-messages, et 3) une approche basée sur l'inférence de langages réguliers pour la grammaire du protocole. Ce n'est que très récemment que les outils ont commencé à nettement se distinguer. Tout d'abord, l'inférence active utilisée par *RolePlayer* et *Netzob* leur permet d'améliorer le découpage en champs. Ensuite, *PRISMA* et *Netzob* prennent en compte l'état du serveur en employant des modèles probabilistes. Enfin, seul l'outil *ReverX* utilise les mêmes techniques pour traiter le format du message et la grammaire du protocole. Cela lui permet notamment de représenter le format des messages sous forme d'automates à états finis alors que les autres outils utilisent généralement de simples templates. Le tableau 4 résume les contributions majeures de chacun des outils ainsi que le défi de la rétro-conception de protocole auquel il répond.

4.2. Inférence applicative

Lim *et al.* sont les premiers à avoir présenté un outil, ***FFE/x86*** (Lim *et al*., 2006), d'inférence applicative de format de messages. Le but de cet outil est d'obtenir le format des données produites par une application en employant une analyse purement statique du binaire. À notre connaissance, il s'agit du seul outil à employer cette approche. Ainsi, il n'est pas soumis au défi d'observabilité rencontré par les outils analysant une trace d'exécution. De plus, il fait partie des rares outils étudiés (avec *Rosetta*, *Replayer* et *Dispatcher*) à inférer le format des messages produits et envoyés par l'application analysée (par opposition aux messages reçus et traités). Plus précisément, il déduit, du graphe d'appel des fonctions, un automate hiérarchique représentant le format des messages. Ensuite, deux techniques d'analyse statique, *VSA (Value Set Analysis)* et *ASI (Agregate Structure Identification)* implémentées dans l'outil *CodeSurfer/x86* (Reps *et al*., 2005), sont utilisées afin d'obtenir un sur-ensemble des valeurs prises par les données en sortie. Le résultat de ces analyses est incorporé directement dans le modèle du format des messages pour le généraliser. La sortie est exprimée sous la forme d'une expression régulière qui est plus facilement exploitable par un analyste humain que l'automate hiérarchique.

En parallèle à ces travaux, Newsome *et al.* ont publié l'outil ***RePlayer*** (Newsome *et al*., 2006). Contrairement à *FFE/x86*, ils utilisent une analyse hors-ligne d'une trace d'exécution. Leur but est de pouvoir rejouer des communications, c'est-à-dire, tout comme *RolePlayer* (Cui *et al*., 2006), d'identifier les champs du protocole dépendant de l'environnement. Dans ce contexte, ils sont les seuls à avoir formalisé le problème de rejeu de communications. Pour cela, ils déduisent des messages, des contraintes identifiant les données liées à l'environnement. Pour le rejeu, ils utilisent la formule de contraintes couplée avec un solver (STP) pour générer les messages qui auraient été produits dans un nouvel environnement. Cependant, seules les contraintes sont déduites et le format des messages n'est pas réellement inféré. Par conséquent, même

Tableau 3. Résumé des outils d'inférence réseau. Un tiret indique que l'outil ne pratique pas ce type d'inférence. Par exemple, Veritas *ne travaille pas sur le format des messages*

Outils	**Approche**	**Classification des messages**	**Format du message**		**Grammaire du protocole**
		Mesure basée sur :	*Dépendances entre champs*	*Modèle utilisé*	*Modèle utilisé*
PI Project	Passive	Alignement + Classification manuelle	Non	Template	-
ScriptGen	Passive	Alignement + classification automatique	Non	Template	Automate fini déterministe
RolePlayer	Active	Alignement	Oui Sans encapsulation	Template	-
Discoverer	Passive	Délimiteurs + classification automatique	Non	Template	-
Veritas	Passive	Découpage récursif en champs de taille fixe + classification automatique	-	-	Automate fini déterministe
ReverX	Passive	Délimiteurs + classification automatique	Non	Automate fini déterministe	Automate fini déterministe
ASAP	Passif	Délimiteurs ou découpage + classification algébrique	Non	Template	-
PRISMA	Passive	Délimiteurs ou découpage + classification automatique	Non	Template	Chaînes de Markov cachées
Netzob	Active	Alignement + inférence active	Oui	Template	Machines de Mealy non déterministes

Tableau 4. Résumé des contributions majeurs de chaque outil d'inférence réseau et du challenge qui est résulo par cette contribution

Outil	**Contribution**	**Défi résolu**
PI Project	Algorithme d'inférence réseau de format de messages basé sur Needleman & Wunch avec du clustering manuel	Découpage automatique en champs
ScriptGen	Inférence de la grammaire de protocole par la théorie des langages, micro/macro clustering	Inférence de la grammaire de protocole + clustering automatique + nettoyage de traces réseau contenant du bruit
Roleplayer	Inférence active des champs dépendant de l'environnement	Rejeu de traces réseau dans un autre environnement
Discoverer	Classification hiérarchique et récursive des messages à partir d'un découpage par délimiteurs connus + heuristiques pour les relations entre champs	Encapsulation des champs + identification de dépendances telles que les tailles ou les champs de session
Veritas	Clustering basé sur l'entête des messages permettant l'inférence de la grammaire	Réduction du volume de données à traiter + inférence de la grammaire sans le format de message
ASAP	Clustering basé sur une métrique algébrique sans inférence de format de message précis	Travail sur de nouvelles métriques pour le clustering des messages
PRISMA	Utilisation de chaines de Markov cachées pour la grammaire de protocole	Représentation d'états ne se traduisant pas forcément par une émission de messages
ReverX	Inférence de format de messages par théorie des langages	Amélioration de la généralisation des messages
Netzob	Grammaire de protocole inférée de façon active + interactions utilisateur	Apprentissage de séquences de messages par approche active
Netzob	Inférence active du format des messages + corrélation temporelle pour les classes de messages	Amélioration de la précision du format des messages, réduction du temps d'apprentissage pour les machines de Mealy

si l'outil accomplit à lui seul cette tâche, il est difficile de généraliser son usage à d'autres domaines. Son incapacité à inférer le format des messages et la grammaire du protocole implique que, contrairement à *ScriptGen*, *Replayer* rejoue exactement la même séquence de classes de messages.

Peu après, Caballero *et al.* ont publié ***Polyglot*** (Caballero *et al*., 2007) qui a une approche différente pour l'inférence des messages basée sur l'analyse des traces d'exécution pour l'inférence du format des messages. Cette approche et les techniques d'analyse associées sont majoritairement reprises par les outils suivants. Par contre, il laisse de côté la classification des messages ainsi que l'inférence de la grammaire du protocole. Il ne fournit donc pas un format de message par classe de messages. Pour ce faire, les données reçues par l'application analysée sont suivis dans la trace d'exécution. Prenons l'exemple où le message est reçu dans un buffer. Avant le début de l'analyse de la trace d'exécution, chaque octet de ce buffer (qui est présent dans la mémoire de l'application) est identifié de manière unique. En effet, la trace d'exécution contient l'état de la mémoire de l'application et il est possible de lui ajouter des méta-données telles que des identifiants. Ces octets sont dit *teintés*. *Polyglot* suit alors les manipulations de cette mémoire par les instructions de la trace. Par exemple, si une instruction copie un octet de la mémoire teintée dans un registre, ce registre est alors teinté. Plus généralement, si l'opérande d'entrée d'une instruction est teintée, alors l'opérande de sortie le sera aussi, on parle de propagation de la teinte. Cela permet d'identifier toutes les instructions manipulant les données reçues par l'application. Cette technique est appelée *Data Tainting Analysis*, elle est utilisée par la majorité des outils d'inférence applicative. Pour obtenir la sémantique des champs, *Polyglot* utilise des heuristiques sur des séquences d'instructions. Globalement, elles correspondent à la recherche dans la trace d'exécution de motifs qui reflètent cette sémantique. Les premières heuristiques les plus importantes se focalisent sur les champs `taille`. Par exemple, une trace d'exécution qui indique que la valeur d'un champ est utilisée pour délimiter un second champ laisse supposer que la valeur premier correspond à la taille du second. La seconde étape permet de retrouver les `délimiteurs` et les `mots-clés` du message. Pour ce faire, si l'instruction de la trace est une comparaison d'une opérande teintée à une valeur fixe, alors l'identifiant de cette opérande teintée est sauvegardé dans une table nommée *table de tokens*. En analysant cette table, les `délimiteurs` et `mots-clés` sont identifiés. Par exemple, les portions successives du message comparées à une valeur fixe sont identifiées comme un seul champ délimité par cette valeur. Pour finir, ces informations sont regroupées pour former le format du message. Cette approche apporte une amélioration de la qualité de l'inférence du format des messages par rapport aux précédentes contributions, notamment celles basées sur l'identification des `mots-clés` ainsi que sur le découpage en champs du message. Cependant, tout comme dans les précédentes contributions, le format du message ne reflète pas l'encapsulation de champs. Les auteurs de *Polyglot* ont identifié une limitation importante à leur outil : si le format du message n'est pas strict, en permettant plusieurs délimiteurs par exemple (HTTP autorise l'usage des espaces ou des tabulations comme séparateurs), alors la trace d'exécution ne corres-

pondra pas aux motifs d'exécution employés dans les heuristiques et cette souplesse du format ne sera pas inférée dans le format du message.

L'outil ***Rosetta*** (Caballero, Song, 2007) est directement issu des travaux de *Polyglot* et de *Replayer*. Son but, tout comme celui de *Replayer* (Newsome *et al*., 2006), est de rejouer des communications dans un autre environnement. Les auteurs étendent l'identification des champs aux messages produits en sortie par l'application en analysant les retours des appels systèmes qui sont utilisés pour construire les messages envoyés. Ils identifient également les champs de type `hash` ou `checksum` même s'ils ne dépendent pas directement de l'environnement. Enfin, ils apportent une nouvelle technique pour identifier les champs dit de `session` correspondant à une dépendance inter-message en regardant si des données d'un message reçu sont utilisées pour construire un des messages envoyés.

Trois outils d'inférence du format des messages, développés en parallèle sont issus des travaux sur *Polyglot* : *AutoFormat*, *Prospex* et *Tupni*. Le premier, ***AutoFormat*** [6] (Lin *et al*., 2008), repose sur l'intuition forte que différents champs d'un message sont traités par des portions de code différentes. Ainsi, le contexte d'exécution associé au traitement de chacun de ces champs est unique. Cet outil découpe donc les messages en fonction de l'ordre dans lesquels les différentes portions du code binaire sont exécutées tout en tenant compte de l'encapsulation. Par construction, le message est découpé suivant un arbre où chaque nœud représente un champ. Ainsi, des champs peuvent contenir des séquences de champs. Enfin, il retrouve les champs parallèles (séparés par un | en notation BNF) qui sont généralement traités dans des portions de code partageant un même historique de contexte d'exécution, c'est-à-dire la même pile d'appels. De par son architecture, cet outil est le seul qui n'a pas besoin d'analyser les boucles. De plus, il est le seul qui pourrait gérer une application qui traite les champs d'un message via des fonctions récursives. Il est à noter que l'on peut distinguer un nouveau défi introduit par cet outil consistant à mesurer la distance entre plusieurs portions de code.

Dans le second outil, ***Prospex*** (Wondracek *et al*., 2008), Wondracek *et al.* présentent dans un premier temps des techniques d'inférence de format de message. Ainsi, ils étendent l'analyse de la *table de tokens* de *Polyglot* (Caballero *et al*., 2007) pour identifier une hiérarchie de champs. De plus, les auteurs s'appuient sur l'algorithme *Needleman & Wunsch* appliqué aux séquences de champs pour généraliser les formats obtenus. Dans un second temps, ces travaux sont poursuivis par Comparetti *et al.* dans (Comparetti *et al*., 2009) pour l'inférence de la grammaire de protocoles. La principale contribution de cet article est le travail sur la classification des messages. Les messages sont regroupés en classes à partir de plusieurs métriques. La première est issue des travaux présentés avec *Discoverer* (Cui, Kannan, Wang, 2007) et concerne l'alignement de séquences de champs qui permet de définir une distance entre deux messages. La seconde métrique mesure la similarité des traces d'exécution de chacun des messages sachant que des messages similaires ont de fortes chances d'être traités

6. Sources disponibles sur demande à l'auteur

par les mêmes portions du code binaire. Enfin, la dernière métrique mesure l'impact des messages sur le système, notamment l'envoi de données sur le réseau, l'ouverture de fichiers, la création de dossier... Ils utilisent la moyenne de ces trois valeurs comme distance entre deux messages, et regroupent les messages par classification. La grammaire du protocole est représentée sous forme d'automate fini déterministe qui est construit à partir de techniques classiques de théorie du langage.

Peu après la publication de *AutoFormat*, Cui *et al.* ont présenté ***Tupni*** (Cui *et al.*, 2008). Cet outil fait suite à leurs travaux sur *Discoverer* (Cui, Kannan, Wang, 2007), où ils rencontraient des difficultés à inférer la sémantique des champs lors de la rétro-conception de format de message par une inférence réseau. Dans un premier temps, l'outil découpe le message en différents champs en fonction des instructions qui accèdent à des portions du message. Dans un second temps, les champs appartenant à une suite, répétée un nombre arbitraire de fois ($\star$ en notation BNF) ou au moins une fois (+ en notation BNF), sont identifiés à partir d'une analyse des boucles du programme. La troisième étape identifie les contraintes sur les champs du message, soit en utilisant les heuristiques de *Polyglot* (valeurs fixes ou champs `taille`) et *Rosetta* (dépendance inter-messages), soit à l'aide de prédicats symboliques similaires à ceux de *Replayer* (dépendances intra-messages). Enfin, *Tupni* généralise ses résultats à travers l'analyse de plusieurs messages. De plus, cet outil peut être couplé à *ShieldGen* (Cui, Peinado *et al.*, 2007) pour la création automatique de signatures d'attaques sur une vulnérabilité inconnue. Il peut aussi faire de l'inférence de format de fichiers en considérant ceux-ci comme des messages reçus.

C'est notamment l'approche utilisée par ***ConfigRE*** (R. Wang *et al.*, 2008) de Wang *et al.* qui considère le fichier de configuration d'une application comme un message reçu par cette application. Ainsi, cet outil peut être étudié comme un outil d'inférence de format de message basée sur l'application. Son but est la rétro-conception de la configuration de contrôle d'accès utilisé par une application, par exemple la configuration des accès aux fichiers d'un serveur apache. Cet outil a des spécificités liées à son domaine d'application. Son approche pour le format de la configuration est classique : il découpe le fichier en champs puis tente d'inférer la sémantique de ces champs. La séparation en champs, tout comme l'outil *Discoverer*, dépend de délimiteurs connus. De plus, du fait du domaine du contrôle d'accès, il introduit les délimiteurs *appairés* tels que : `{` et `}` ou `(` et `)` ou de façon plus complexe `<Directory>` et `</Directory>`. Les auteurs utilisent une heuristique spécifique d'analyse des sauts conditionnels pour retrouver ces éléments. De plus, ils analysent l'interaction entre la teinte du fichier de configuration lors de son chargement en mémoire et la teinte de différentes requêtes de tests pour inférer la sémantique des champs. Cet outil utilise une approche active de l'inférence en choisissant spécifiquement les requêtes à effectuer. Ces informations permettent de construire un arbre sémantique du fichier de configuration. Enfin, il modifie les champs du fichier de configuration dont la sémantique est `permission` avec une valeur aléatoire, puis lors de l'analyse de la nouvelle trace d'exécution, identifie toutes les valeurs légitimes que peut prendre ce champs, l'application testant successivement ces valeurs légitimes contre celle de la configuration avant de sortir en échec.

Alors que les précédents outils étaient limités à l'analyse de protocoles non chiffrés, ***ReFormat*** (Z. Wang *et al*., 2009) de Wang *et al.* s'attache à inférer le format de messages chiffrés. Pour cela, il doit dans un premier temps retrouver automatiquement les fonctions de chiffrement et dans la trace d'exécution, une zone de la mémoire où le message est déchiffré. Ainsi, il retrouve les fonctions possédant un grand nombre d'instructions arithmétiques qui sont généralement des fonctions cryptographiques. Dans leur outil, les auteurs utilisent la notion de *durée de vie des données* (Chow, 2006) pour identifier les sorties des fonctions cryptographiques correspondant au message en clair. Enfin, ils couplent leur outil à *AutoFormat* (Lin *et al*., 2008) pour faire automatiquement l'inférence du format du message. Cependant, *ReFormat* ne permet d'inférer que le format des messages reçus chiffrés par une application, pas celui des messages produits par celle-ci.

Caballero *et al.* ont poursuivi leurs travaux de *Polyglot* et *Rosetta* en incorporant des évolutions apparues dans *Tupni*, *AutoFormat*, *Prospex* et *ReFormat*. Il en découle l'outil ***Dispatcher*** [7] (Caballero *et al*., 2009 ; Caballero, Song, 2013). Ainsi, cet outil permet de retrouver l'encapsulation des champs en utilisant une analyse plus poussée de la *table de tokens* présentée dans *Polyglot*. Les différentes sémantiques de champs inférées ont aussi été élargies. Ces travaux sont détaillés dans la thèse de J. Caballero (Caballero Bayerri, 2010) de même qu'une formalisation plus poussée du format des messages inférés. Si cette formalisation est la plus poussée dans cette thèse, de notre point de vue elle souffre de la non-distinction des champs qui sont utiles à la transmission des messages (`taille, offset, hash, checksum, délimiteurs...`) de ceux qui sont utiles à l'analyse (`port, adresse IP, timestamp, nom de fichier...`). Ces derniers dépendent directement du domaine d'application de l'inférence alors que les premiers sont communs à toutes les inférences. Le but de *Dispatcher* est de permettre l'infiltration de botnets. Or, dans de nombreux cas, les malwares utilisent du chiffrement et de l'obfuscation. Ainsi, Caballero *et al.* étendent les techniques introduites dans *Reformat* (Z. Wang *et al*., 2009) d'identification des fonctions de chiffrement et des messages en clair au cas du bot *MegaD*. Cela lui a permis de traiter les défis suivants : multiples points de déchiffrement, déchiffrement de portions du message à la volée, identification des fonctions de chiffrement des données produites. Néanmoins, l'usage de *Dispatcher* est limité par l'absence de travail sur la classification des messages et de l'absence d'inférence de la grammaire du protocole. L'article (Caballero *et al*., 2009) ne contient pas les informations sur la réalisation de ces travaux.

À notre connaissance, ***MACE*** (Cho *et al*., 2010 ; 2011) est le seul outil à pratiquer de l'inférence de grammaire de protocoles par une exécution concolique (concrète et symbolique) de l'application. L'exécution concolique permet d'explorer les différentes valeurs que peuvent prendre les messages du protocole et construire sa grammaire. Dans un premier temps, les analystes devaient fournir les fonctions d'abstraction des messages d'entrée et de sortie, mais ce problème a été partiellement résolu

7. Sources disponibles sur demande à l'auteur

dans (Cho *et al*., 2011), où seule l'abstraction des messages de sortie est maintenant requise. Cet outil utilise l'algorithme L^* (Angluin, 1987) pour inférer la grammaire du protocole. Les auteurs ont utilisé leur outil sur le protocole du botnet *MegaD*, ce qui leur a permis de découvrir des portions du protocole qui n'étaient pas observables lors des phases d'analyses classiques. Si cet outil ne permet pas d'inférer le format des messages mais juste leur type, il fournit cependant des instances de messages concrètes pour arriver à un certain niveau de l'exécution. Cela est utile pour valider l'exploitabilité d'une vulnérabilité. Il a également permis de détecter des vulnérabilités dans différentes applications (trois dans Samba 3.3.4, trois dans Vino 2.26.1 et une dans RealVNC 4.1.2).

Suite aux travaux d'inférence de format du message, certaines techniques ont été utilisées pour l'inférence de structures de données, c'est notamment le cas des outils *REWARDS*, *Howard*, *ARTISTE* et *ARGOS*. Ainsi, Lin *et al.* sont les premiers à avoir adapté les techniques d'inférence de format de message à l'inférence de structure de données dans l'outil ***REWARDS*** (Lin *et al*., 2010 ; Lin, 2011). Il permet de pratiquer des *analyses forensics de la mémoire* ainsi que du fuzzing. Pour ce faire, ils utilisent la même analyse de trace d'exécution que dans *Dispatcher* (Caballero *et al*., 2009) pour retrouver le format des structures de données (correspondant à des messages pour *Dispatcher*) et la sémantique des champs. Cependant, ils doivent aussi pouvoir analyser la mémoire à un moment spécifique de l'exécution. Ainsi, ils étendent leur analyse en ajoutant une analyse en arrière de la trace d'exécution pour retrouver d'où proviennent les éléments en mémoire.

Le second outil, ***Howard*** (Slowinska *et al*., 2010 ; 2011) de Slowinska *et al.*, a été conçu pour aider le débogage d'applications binaires. Il enregistre les buffers alloués dynamiquement, puis identifie la manière dont ils sont accédés pour déterminer leur découpage en différents champs. Cette étape est similaire aux travaux dans *FFE/x86* (Lim *et al*., 2006) ou *Tupni* (Cui *et al*., 2008). Cependant, son analyse dynamique lui permet une plus grande précision des résultats. En effet, il n'y a pas de sur-approximation des valeurs prises par les champs de la structure. De plus, il inclut une amélioration de l'inférence de tableaux présentée dans *REWARDS* et *Polyglot* (Caballero *et al*., 2007) et il propage de manière complémentaire à *REWARDS* les structures issues de fonctions connues. Pour finir, en plus d'aider au débogage d'application binaire, *Howard* peut détecter et empêcher des corruptions en mémoire en instrumentant l'application d'origine, parfois de manière manuelle, pour sécuriser les déréférencements de pointeurs.

Peu après, en collaboration avec les auteurs de *REWARDS*, Caballero *et al.* ont développé ***ARTISTE*** (Caballero *et al*., 2012). Cet outil étend des techniques présentées dans *Dispatcher* (Caballero *et al*., 2009) et dans *REWARDS* (Lin, 2011) pour l'inférence de structures et de format de messages. Ainsi, il peut inférer des structures complexes telles que des arbres ou des listes doublement chaînées. Il présente également une nouvelle technique permettant d'identifier des structures similaires allouées à différents endroits du programme. Enfin, il améliore la vitesse d'inférence des structures simples.

Enfin, Zeng et Lin ont développé ***ARGOS*** (Zeng, Lin, 2015) pour aider à la compréhension et au monitoring des noyaux des systèmes d'exploitation. Ils utilisent les précédentes techniques présentées dans *ARTISTE* pour propager l'inférence de sémantique sur les structures du noyau. De plus, ils utilisent le contexte d'exécution, c'est-à-dire les actions faites par le noyau telles que la création de processus ou la lecture de fichiers, pour classifier les différents objets du noyau, et leur donner un sens compréhensible par un humain.

Synthèse

Le tableau 5 résume les différentes approches d'inférence de protocole basée sur une analyse d'une application et le tableau 6 identifie les outils pratiquant l'inférence de structures de données. L'approche statique ou dynamique et active ou passive est résumée en seconde colonne. Pour le tableau 5, la troisième colonne met en évidence les outils pratiquant le regroupement de message pour inférer la grammaire de protocole (dernière colonne). Ces modèles sont classés suivant leur niveau de représentativité, rappelons que ces modèles sont brièvement présentés en section 3.3. Presque tous ces outils infèrent le format des messages reçus. Cependant, peu d'entre eux sont capable d'inférer le format des messages produits par une application. La colonne `généralisation du modèle` permet d'identifier les outils capables de généraliser le format de message, soit en obtenant directement un format générique (*FFE/x86*), soit en analysant plusieurs messages. Pour les structures de données (tableau 6) le type de structures identifiée est important, ainsi, deux types de structures sont identifiées, les structures simples (références directes) et les structures récursives (listes chainées, arbres...). Ces tableaux mettent en évidence que *FFE/X86* est le seul à employer une analyse statique du binaire, alors que les autres outils utilisent une approche dynamique, dont la majorité de manière passive. De plus, les outils s'intéressent principalement au format du message et négligent la classification de ces messages, ce qui constitue une limite à l'inférence de la grammaire du protocole. En conséquence, seuls *Prospex* et *MACE* qui classent les messages, permettent d'inférer la grammaire du protocole. Nous constatons également une ouverture des outils vers des perspectives nouvelles avec leur adaptation à l'inférence de format de fichiers (*Tupni* et *ConfigRE*) et à l'inférence de structures de données, notamment pour le noyau (*REWARDS*, *Howard*, *ARTISTE* et *ARGOS*). Les tableaux 7 et 8 résument les contributions majeures de chacun des outils ainsi que le défi de la rétro-conception de protocole auquel il répond.

5. Conclusion

Dans cet état de l'art de la rétro-conception de protocole, nous avons dans un premier temps explicité les termes les plus couramment utilisé pour ce domaine. Nous avons ensuite présenté les différentes étapes nécessaires à l'inférence de protocole. Ces étapes sont souvent complexes et présentent des défis qui ont été identifiés dans la section suivant. Ces étapes et défis sont souvent laissés de coté dans la littérature pour s'intéresser à l'étape finale de la rétro-conception de protocole, l'inférence de format

Tableau 5. Résumé des outils d'inférence applicative. Un tiret indique que l'outil ne traite pas cet aspect de l'inférence de protocole

Outils	**Méthode d'analyse**	**Classification des messages** *Mesure basée sur :*	**Format du message** *Reçus Envoyés*	*Modèle utilisé*	*Généralisation du modèle*	**Grammaire du protocole** *Modèle utilisé*
FFE/x86	Statique	-	Envoyés	Expression régulière	Oui	-
Replayer	Active dynamique	-	Reçus Envoyés	Template	Non	-
Polyglot	Passive dynamique	-	Reçus	Template annoté	Non	-
Rosetta	Passive dynamique	-	Reçus Envoyés	Template annoté	Non	-
AutoFormat	Passive dynamique	-	Reçus	Arbre annoté	Oui	-
Prospex	Passive dynamique	Alignement + comportement de l'application	Reçus	Arbre annoté	Oui	Automate fini déterministe
Tupni	Passive dynamique	-	Reçus	Arbre annoté	Oui	-
ConfigRE	Active dynamique	-	Reçus	Arbre annoté	Oui	-
ReFormat	Passive dynamique	-	Reçus	-	-	-
Dispatcher	Passive dynamique	-	Reçus Envoyés	Arbre annoté	Non	-
MACE	Active concolique	Abstraction manuelle des messages produits	-	-	-	Machines de Mealy

Tableau 6. Résumé des outils d'inférence de structures. Un tiret indique que l'outil ne traite pas cet aspect de l'inférence

Outils	**Méthode d'analyse**	**Classification des structures** *Mesure basée sur :*	**Inférence de la structure**	
			Type de structure	*Généralisation du modèle utilisé*
REWARDS	Passif dynamique	Contexte d'exécution	Simple	Non
Howard	Passif dynamique	-	Simple	Non
ARTISTE	Passif dynamique	Contexte d'exécution	Récursive	Oui
ARGOS	Passif dynamique	-	Récursive	Non

de message ou de grammaire de protocole. Alors, nous avons présenté l'évolution des outils d'inférence de protocole, les classifiant aussi suivant leur approche active ou passive.

À la lumière des différents outils que nous avons étudiés, nous pouvons faire le constat que ces dernières années, beaucoup d'outils de rétro-conception de protocole ont vu le jour. Ils traitent de plus en plus de domaines applicatifs, allant jusqu'à inférer les structures de données complexes dans les travaux les plus récents. Alors que les premières étapes de pré-traitement nécessitent encore un investissement important de l'analyste, l'étape d'inférence en elle-même est de plus en plus automatisée et ne nécessite pratiquement plus d'intervention humaine. En ce qui concerne les pistes de travaux futurs, nous constatons que la recherche est encore assez focalisée sur l'inférence du format des messages. En effet, il existe peu d'outils d'inférence de la grammaire du protocole, et ceux-ci sont majoritairement basés sur une inférence réseau. Cela s'explique par un volume important de données complexes à traiter lors de l'inférence de grammaire basée sur l'analyse de l'application. De plus, certains formats de messages et grammaires de protocoles peuvent être très complexes en raison de fortes dépendances entre les champs et d'importantes possibilités d'encapsulation. Or, globalement, les modèles utilisés pour inférer les formats et les grammaires sont équivalents aux automates finis et ils peuvent ne pas être assez expressifs pour représenter toutes les subtilités des formats de messages ou des échanges.

Dans cet article, nous n'avons pas pu effectuer d'évaluation des performances des différents outils. Cela est dû notamment à la non disponibilité de la majorité d'entre eux (sauf *Netzob*, *ReverX*, *PRISMA*, *PI Project*) et seul *Netzob* est encore maintenu à ce jour. Par ailleurs, une comparaison en terme de temps/mémoire est difficile car ces outils sont développés dans différents langages de programmation (interprété (python), compilé (C)). De plus, certains outils ne s'attaquent qu'à un aspect précis de la rétro-conception (*PRISMA* n'obtient pas de format de message précis, *Veritas*, pas du

Tableau 7. Résumé des contributions majeurs de chaque outil d'inférence applicative et du challenge qui est résolu par cette contribution

Outil	Contribution	Défi résolu
FFE/x86	Analyse statique du binaire de l'application à partir d'un point de synchronisation	format des données générées par une application sans exécution
RePlayer	Rejeu de traces par obtention des contraintes liées à l'environnement	Rejeu de traces dans un autre environnement
Polyglot	Analyse de la trace d'exécution d'une application pour inférer le format des messages reçus	Inférence du format des messages reçus par l'application
Rosetta	Extension des travaux de *Polyglot* aux messages produits : analyse des retours d'appels système	Inférence des messages produits par l'application
AutoFormat	Découpage en champs basé sur la région d'exécution du code	Traitement d'un message par une fonction récursive + nouvelle technique de découpage de message en champs
Prospex	Généralisation par Needleman & Wunsch du format de message + analyse du système servant de métrique pour le clustering de messages	Généralisation du format de message + nouvelles métriques pour le clustering
Tupni	Intégration des travaux précédents + généralisation par analyse de multiples messages	Généralisation du format des messages
ConfigRE	Analyse dynamique active du traitement du fichier de configuration	Inférence de délimiteurs appairés dans un fichier de configuration
ReFormat	Identification automatique des interfaces de chiffrement par analyse statistique des instructions arithmétiques	Inférence de messages chiffrés
Dispatcher	Extension *ReFormat* aux messages générés + formalisation du modèle de format de message	Défi concret d'un botnet : multiples interfaces de chiffrement, pas d' accès au serveur de contrôle
MACE	Inférence concolique	Exploration automatique des chemins de traitement des messages reçus par l'application

Tableau 8. Résumé des contributions majeurs de chaque outil d'inférence de structures de données et du challenge qui est résolu par cette contribution

Outil	Contribution	Défi résolu
REWARDS	Inférence de structures de données par analyse montante à partir des arguments d'appels système prototypés par l'utilisateur	Analyse forensic d'une trace d'exécution
Howard	Typage automatique du format des structures simples allouées dynamiquement	Typage automatique des structures allouées dynamiquement
ARTISTE	Analyse des déréférencements de pointeurs dans une trace d'exécution	Inférence des structures de données complexes : listes chainées, arbres, tableaux
ARGOS	Analyse des actions du système pour la classification automatique des structures allouées dynamiques dans le noyau Linux	Aider un humain à comprendre et analyser les structures du noyau et comment elles sont accédées

tout, mais fait quand même du clustering). Le corpus diffère complètement suivant les approches, ainsi *FFE/x86* et *MACE* n'ont besoin d'aucun corpus, les autres outils d'inférence applicative ont besoin d'au moins un message, alors que *Discoverer* utilise un corpus de plus de 500 000 messages. Une grande majorité des outils étudiés reposent à un moment où un autre sur des techniques de clustering des messages. Cependant, l'étape de clustering des messages est pratiquée par la majorité des outils, avec des entrées à cette étape similaires (un corpus de message). Nous pensons qu'une étude comparative de ces différentes techniques de clustering bénéficierait grandement au domaine de la rétro-conception de protocole.

L'état de l'art proposé dans cet article met en évidence que si les différents travaux sur l'inférence de protocole ont peu à peu traité les défis qui y sont associés, de nouvelles techniques ont parfois introduit de nouveaux défis à résoudre pour le futur. Par exemple, l'outil *AutoFormat* effectue le découpage en champs du message en mesurant la similarité des portions de code traitant des parties du message. L'outil *Prospex* utilise la similarité des portions de code comme métrique de classification des messages. Ces deux outils ont ainsi introduit un défi intéressant concernant la mesure de similarité dans le code binaire, ils seront d'autant plus efficace que ces mesures de similarités seront précises.

Par ailleurs nous pouvons identifier d'autres axes de recherches qui pourraient être pertinents pour la rétro-conception de protocoles. Par exemple, *Dispatcher* (Caballero, Song, 2013) et *ReFormat* (Z. Wang *et al*., 2009) doivent faire face à l'identification des données non chiffrées lors de l'analyse des messages. D'autres approches basées sur

la mesure de l'entropie des données auraient pu être utilisées et fournir des résultats intéressants. Ce défi s'apparente au problème du placement des sondes présenté en section 3.1, mais le but ici est de déterminer automatiquement où placer une sonde de capture de messages dans une application binaire.

Bibliographie

Angluin D. (1987). Learning regular sets from queries and counterexamples. *Information and Computation*, vol. 75, n° 2, p. 87–106.

Antunes J., Neves N., Verissimo P. (2011). Reverse Engineering of Protocols from Network Traces. In *2011 18th Working Conference on Reverse Engineering (WCRE)*, p. 169–178. New York, NY, USA, IEEE.

Beddoe M. (2004a). *Network Protocol Analysis using Bioinformatics Algorithms.* http://www.4tphi.net/~awalters/PI/pi.pdf.

Beddoe M. (2004b). *Protocol Informatics Project.* http://www.4tphi.net/~awalters/PI/PI.html.

Bohlin T., Jonsson B. (2008). *Regular Inference for Communication Protocol Entities*. Technical Report n° 2008-024. Uppsala University, Sweden, Department of Information Technology, Uppsala University.

Bossert G. (2014). *Exploiting Semantic for the Automatic Reverse Engineering of Communication Protocols.* phdthesis, Supélec.

Bossert G., Guihery F., Hiet G. (2014, jun). Towards automated protocol reverse engineering using semantic information. In *Proceedings of the 9th ACM symposium on Information, computer and communications security*, p. 51–62. Kyoto, Japan, ACM.

Bossert G., Hiet G., Henin T. (2011). Modelling to Simulate Botnet Command and Control Protocols for the Evaluation of Network Intrusion Detection Systems. In *2011 Conference on Network and Information Systems Security (SAR-SSI)*, p. 1–8. La Rochelle, France, IEEE.

Caballero J., Grieco G., Marron M., Lin Z., Urbina D. (2012). *ARTISTE: Automatic Generation of Hybrid Data Structure Signatures from Binary Code Executions*. Technical Report n° TR-IMDEA-SW-2012-001. Madrid, Spain, IMDEA Software Institute.

Caballero J., Poosankam P., Kreibich C., Song D. (2009). Dispatcher: enabling active botnet infiltration using automatic protocol reverse-engineering. In *Proceedings of the 16th ACM conference on Computer and communications security*, p. 621–634. New York, NY, USA, ACM.

Caballero J., Song D. (2007). *Rosetta: Extracting protocol semantics using binary analysis with applications to protocol replay and NAT rewriting*. Technical Report n° CMU-CyLab-07-014. Pittsburgh, USA, Carnegie Mellon University.

Caballero J., Song D. (2013, février). Automatic protocol reverse-engineering: Message format extraction and field semantics inference. *Computer Networks*, vol. 57, n° 2, p. 451–474.

Caballero J., Yin H., Liang Z., Song D. (2007). Polyglot: automatic extraction of protocol message format using dynamic binary analysis. In *Proceedings of the 14th ACM conference on Computer and communications security*, p. 317–329. New York, NY, USA, ACM.

Caballero Bayerri J. (2010). *Grammar and model extraction for security applications using dynamic program binary analysis*. Thèse de doctorat non publiée, Carnegie Mellon University, Pittsburgh, PA, USA.

Cho C. Y., Babić D., Poosankam P., Chen K. Z., Wu E. X., Song D. (2011, aug). MACE: model-inference-assisted concolic exploration for protocol and vulnerability discovery. In *Proceedings of the 20th USENIX conference on Security*, p. 19. Berkeley, CA, USA, USENIX Association.

Cho C. Y., Babić D., Shin E. C. R., Song D. (2010). Inference and Analysis of Formal Models of Botnet Command and Control Protocols. In *Proceedings of the 17th ACM Conference on Computer and Communications Security*, p. 426–439. New York, NY, USA, ACM.

Chow J. (2006). *Understanding Data Lifetime*. Thèse de doctorat non publiée, Stanford University, Stanford, CA, USA. (AAI3219247)

Comparetti P., Wondracek G., Kruegel C., Kirda E. (2009). Prospex: Protocol Specification Extraction. In *2009 30th IEEE Symposium on Security and Privacy*, p. 110–125. Berkeley, USA, IEEE.

Cui W., Kannan J., Wang H. J. (2007). Discoverer: automatic protocol reverse engineering from network traces. In *Proceedings of 16th USENIX Security Symposium on USENIX Security Symposium*, p. 14:1–14:14. Berkeley, CA, USA, USENIX Association.

Cui W., Paxson V., Weaver N., Katz R. H. (2006, February). Protocol-independent adaptive replay of application dialog. In *Proceedings of the 13th annual network and distributed system security symposium (ndss)*. San Diego, USA, Internet Society. Consulté sur http://research.microsoft.com/apps/pubs/default.aspx?id=153197

Cui W., Peinado M., Chen K., Wang H. J., Irun-Briz L. (2008). Tupni: automatic reverse engineering of input formats. In *Proceedings of the 15th ACM conference on Computer and communications security*, p. 391–402. New York, NY, USA, ACM.

Cui W., Peinado M., Wang H., Locasto M. (2007). ShieldGen: Automatic data patch generation for unknown vulnerabilities with informed probing. In *IEEE symposium on security and privacy, 2007. SP '07*, p. 252–266. Oakland, USA, IEEE.

Guihery F., Bossert G. (2012a, décembre). The future of protocol reversing and simulation applied on ZeroAccess. In *29c3: 29th chaos communication congress '12*. Hambourg, Germany, C-3.

Guihery F., Bossert G. (2012b). Netzob : un outil pour la rétro-conception de protocoles de communication. In *Symposium sur la sécurité des technologies de l'information et de la communication*. Rennes, France, SSTIC.

Krueger T., Gascon H., Krämer N., Rieck K. (2012). Learning Stateful Models for Network Honeypots. In *Proceedings of the 5th ACM Workshop on Security and Artificial Intelligence*, p. 37–48. New York, NY, USA, ACM. Consulté sur http://doi.acm.org/10.1145/2381896.2381904

Krueger T., Krämer N., Rieck K. (2010). ASAP: Automatic Semantics-Aware Analysis of Network Payloads. In C. Dimitrakakis, A. Gkoulalas-Divanis, A. Mitrokotsa, V. S. Verykios, Y. Saygin (Eds.), *Privacy and Security Issues in Data Mining and Machine Learning*, p. 50–63. Barcelona, Spain, Springer Berlin Heidelberg. (DOI: 10.1007/978-3-642-19896-0_5)

Leita C. (2008). *SGNET : automated protocol learning for the observation of malicious threats*. Thèse de doctorat non publiée, Université de Nice. Consulté sur http://www.eurecom.fr/publication/2709

Leita C., Mermoud K., Dacier M. (2005). ScriptGen: an automated script generation tool for Honeyd. In *Computer Security Applications Conference, 21st Annual*, p. 12 pp.–214. Tucson, USA, IEEE.

Li X., Chen L. (2011). A Survey on Methods of Automatic Protocol Reverse Engineering. In *2011 Seventh International Conference on Computational Intelligence and Security (CIS)*, p. 685–689. Hainan, China, IEEE.

Lim J., Reps T., Liblit B. (2006). Extracting Output Formats from Executables. In *13th Working Conference on Reverse Engineering, 2006. WCRE '06*, p. 167–178. Benevento, Italy, IEEE.

Lin Z. (2011). *Reverse Engineering of Data Structures from Binary*. Thèse de doctorat non publiée, PURDUE UNIVERSITY, West Lafayette, Indiana.

Lin Z., Jiang X., Xu D., Zhang X. (2008). Automatic Protocol Format Reverse Engineering through Context-Aware Monitored Execution. In *Proceedings of the 15th annual network and distributed system security symposium (ndss)*. San Diego, USA, Internet Society.

Lin Z., Zhang X., Xu D. (2010). Automatic Reverse Engineering of Data Structures from Binary Execution. In *Proceedings of the 17th annual network and distributed system security symposium (ndss)*. San Diego, USA, Internet Society.

Narayan J., Shukla S. K., Clancy T. C. (2015). A survey of automatic protocol reverse engineering tools. *ACM Computing Surveys (CSUR)*, vol. 48, n° 3, p. 40.

Needleman S. B., Wunsch C. D. (1970). A general method applicable to the search for similarities in the amino acid sequence of two proteins. *Journal of Molecular Biology*, vol. 48, n° 3, p. 443–453.

Nei M., Tajima F., Tateno Y. (1983). Accuracy of estimated phylogenetic trees from molecular data. *Journal of Molecular Evolution*, vol. 19, n° 2, p. 153–170.

Newsome J., Brumley D., Franklin J., Song D. (2006). Replayer: automatic protocol replay by binary analysis. In *Proceedings of the 13th ACM conference on Computer and communications security*, p. 311–321. New York, NY, USA, ACM.

Reps T., Balakrishnan G., Lim J., Teitelbaum T. (2005). A Next-Generation Platform for Analyzing Executables. In *3rd asian symposium on programming languages and systems*, p. 212–229. Tsukuba, Japan, Springer-Verlag Berlin.

Samba Team. (2017). *Opening windows to a wider world.* http://www.samba.org.

Slowinska A., Stancescu T., Bos H. (2010). *Dynamic data structure excavation*. Technical Report n° IR-CS-55. Amsterdam, Danemark, Vrije Universiteit Amsterdam.

Slowinska A., Stancescu T., Bos H. (2011). Howard: A Dynamic Excavator for Reverse Engineering Data Structures. In *Proceedings of the 18th annual network and distributed system security symposium (ndss)*. San Diego, USA, Internet Society.

Wang R., Wang X., Zhang K., Li Z. (2008). Towards Automatic Reverse Engineering of Software Security Configurations. In *Proceedings of the 15th ACM Conference on Computer and Communications Security*, p. 245–256. Limerick, Ireland, ACM.

Wang Y., Zhang Z., Guo L. (2010, septembre). Inferring Protocol State Machine from Real-World Trace. In S. Jha, R. Sommer, C. Kreibich (Eds.), *Recent Advances in Intrusion Detection*, p. 498–499. Ottawa, Canada, Springer Berlin Heidelberg. (DOI: 10.1007/978-3-642-15512-3_32)

Wang Y., Zhang Z., Yao D. D., Qu B., Guo L. (2011, jan). Inferring Protocol State Machine from Network Traces: A Probabilistic Approach. In J. Lopez, G. Tsudik (Eds.), *Applied Cryptography and Network Security*, p. 1–18. Nerja, Spain, Springer Berlin Heidelberg.

Wang Z., Jiang X., Cui W., Wang X., Grace M. (2009, janvier). ReFormat: Automatic Reverse Engineering of Encrypted Messages. In M. Backes, P. Ning (Eds.), *Computer Security – ESORICS 2009*, p. 200–215. Saint Malo, France, Springer Berlin Heidelberg.

Wondracek G., Comparetti P. M., Krügel C., Kirda E. (2008). Automatic network protocol analysis. In *Proceedings of the 15th annual network and distributed system security symposium (ndss)*. San Diego, USA, Internet Society.

Zeng J., Lin Z. (2015, novembre). Towards Automatic Inference of Kernel Object Semantics from Binary Code. In *18th International Symposium, RAID 2015*, vol. 9404, p. 538–561. Kyoto, Japan, Springer.