# Modelling to Simulate Botnet Command and Control Protocols for the Evaluation of Network Intrusion Detection Systems.

Georges Bossert
AMOSSYS-SUPELEC, Rennes
georges.bossert@amossys.fr

Guillaume Hiet
SUPELEC, Rennes
guillaume.hiet@supelec.fr

Thibaut Henin
AMOSSYS, Rennes
thibaut.henin@amossys.fr

***Abstract*—The purpose of this paper is the modelization and simulation of zombie machines for the evaluation of Network Intrusion Detection Systems (NIDS), used to detect botnets. We propose an automatic method to infer zombies behaviours through the analysis of messages exchanged with their masters. Once computed, a model provides a solution to generate realistic and manageable traffic, which is mandatory for an NIDS evaluation. We propose to use a Stochastic Mealy Machine to model zombies behaviour, and an active inference algorithm to learn it. With our approach, it is possible to generate a realistic traffic corresponding to the communications of botnets while ensuring its controllability in the context of an NIDS evaluation.**

## I. Introduction

Ces dernières années, les systèmes de détection d'intrusion réseau (NIDS[1]) sont de plus en plus sollicités pour la détection des réseaux de zombies [**?**], [**?**]. Il est donc nécessaire d'évaluer les qualités de détection de ces systèmes, en ce qui concerne la menace particulière que constituent les réseaux de zombies (botnet).

La mise en place d'une méthodologie d'évaluation applicable au domaine de la détection de réseau de zombies est conditionnée au respect de plusieurs critères assurant les garanties d'une expérience scientifique rigoureuse. Parmi ces critères, nos travaux adressent notamment les deux critères suivants :

**Reproductibilité :** se réfère à la possibilité, pour une personne indépendante et extérieure au processus orginial, de réaliser de nouveau l'expérimentation et d'obtenir des résultats similaires à l'expérimentation initiale.

**Exactitude :** traduit le degré de concordance entre une valeur dite « de référence » ou considérée comme véritable avec la valeur obtenue par l'évaluation.

De ces deux critères génériques découlent deux conditions : le réalisme et la contrôlabilité de l'expérience. Le réalisme de l'évaluation assure l'exactitude des résultats. La contrôlabilité assure la reproductibilité, en nécessitant une caractérisation complète de l'évaluation [**?**].

L'évaluation d'un NIDS nécessite, entre autres, l'utilisation d'un trafic réseau représentatif de ses conditions opérationnelles. Parmi les différentes méthodes utilisées pour générer du trafic réseau, il est possible de distinguer les trois familles principales suivantes : le rejeu, la génération synthétique et la génération hybride.

**La méthode du rejeu** consiste à utiliser un trafic préalablement capturé pour être ensuite rejoué. Cette méthode est souvent utilisée par un évaluateur pour sa facilité à reproduire un résultat. Cependant, elle nécessite, pour être exacte, de caractériser complètement l'ensemble des évènements présents dans la capture. Cette opération coûteuse, en temps et en expertise [**?**], est nécessaire afin de ne pas introduire de comportements hors-périmètres, tels que des attaques non caractérisées ou le rejeu d'un protocole incompatible avec le NIDS. À cette problématique, se rajoute le droit au respect de la vie privée des acteurs impliqués dans les traces capturées, qui peuvent ne pas souhaiter voir leurs identités réseaux utilisées et publiées.

**La méthode de génération synthétique** permet de pallier à ces problèmes. Cette solution repose sur la génération complète d'un trafic en se basant sur des heuristiques et des statistiques généralement admises [**?**] pour modéliser l'évolution du comportement d'un acteur dans le temps. Cependant, le trafic généré est souvent trop simple et souffre d'un manque de réalisme.

**La méthode hybride** est une solution de génération de trafic réseau qui respecte à la fois les critères de réalisme et de contrôlabilité [**?**]. Contrairement à la génération synthétique, la génération hybride inférère les caractéristiques à partir d'un enregistrement réseau préalablement capturé, pour les ré-utiliser au sein d'un processus de génération synthétique. Cette fois-ci, le trafic généré représente de manière précise la réalité, du moins, à la hauteur des caractéristiques utilisées. La condition de contrôlabilité est également satisfaite car la méthode implique la génération du trafic et non un simple rejeu. Cette dernière approche est celle que nous mettons œuvre dans nos travaux.

Le reste de cet article est organisé comme suit. Dans la section II, nous présentons les botnets et les travaux s'attachant à les détecter. Ensuite, dans la section III, nous introduisons une extension stochastique de la machine de Mealy que nous utilisons pour modéliser le comportement

[1]en anglais, Network Intrusion Detection System.

d'un zombie. Dans la section [IV], nous décrivons notre méthode d'apprentissage du modèle. Enfin, nous présentons l'implémentation et les résultats de notre architecture dédiée à l'évaluation des mécanismes de détection de botnet (section V).

## II. Travaux antérieurs

### A. Les botnets

Un botnet est un ensemble de systèmes compromis, sous le contrôle d'un propriétaire et/ou de plusieurs systèmes de commandes dénommés maîtres du botnet. Les systèmes ou hôtes compromis, aussi appelés zombies, échangent des informations et des ordres avec leurs maîtres via un canal de communication appelé *C&C* pour « *Command and Control* ». Certaines architectures de botnet distribuées [**?**] introduisent la possibilité qu'un zombie puisse communiquer directement avec un autre zombie. Afin de ne pas limiter nos travaux aux seules architectures non-distribuées, nous définissons le zombie à l'origine directe du message comme le maître relatif du zombie qui doit effectuer la commande et/ou répondre à l'ordre. Nous ne différencions pas un maître relatif du véritable propriétaire du réseau.

Le maître communique sur le canal *C&C* pour contrôler les zombies de son réseau et leur assigner une série d'opérations malveillantes à exécuter. Les résultats de l'exécution de ces opérations lui sont ensuite renvoyés. Parmi celles-ci, on peut citer l'envoi massif d'emails, l'exécution d'attaques décentralisées (DDOS) ainsi que des techniques de récupération d'informations personnelles sur les systèmes compromis.

De nos jours, une grande partie des botnets déployés utilise pour leur C&C, des protocoles réseaux comme IRC (Internet Relay Chat), XMPP (Extensible Messaging Protocol and Presence Protocol), l'envoi d'emails, et même des services de réseaux sociaux comme Twitter[2]. Nous avons fait le choix d'appliquer notre solution au protocole IRC. Il s'agit en effet d'un des protocoles les plus utilisés d'après les statistiques annuelles publiées par Shadow Server[3]. Toutefois, ce choix n'est pas restrictif et la méthode s'adapte facilement à d'autres protocoles.

Nos travaux adressant la problématique de l'évaluation des mécanismes de détection de botnet, nous présentons par la suite les principales solutions existantes.

### B. Détection de botnet

Les méthodes de détection de botnet s'organisent en trois grandes familles.

La première regroupe les méthodes de détection se focalisant sur l'impact d'un zombie sur un système. Pour se faire, le code exécutable du zombie est analysé pour comprendre son fonctionnement et établir des signatures virales [**?**], [**?**]. Ces codes sont souvent collectés par des pôts de miels dédiés [**?**], [**?**].

La deuxième famille rassemble l'ensemble des mécanismes consistant à caractériser la prolifération des zombies ainsi que leur topologie réseau. Parmi ces travaux, ceux de Hyunsang et al. [**?**] s'orientent notamment vers l'identification de botnets via la modélisation et l'analyse de communautés.

La dernière famille considère la présence d'un canal de communication de type *C&C* comme le symptôme d'une infection [**?**], [**?**], [**?**]. Cette dernière méthode apparaît comme la plus intéressante puisqu'elle cible le point faible d'un botnet, à savoir son système de communication.

En effet, comme le montrent les travaux de Guoefei Gu [**?**], la première étape d'une lutte efficace contre la prolifération des botnets consiste à détecter leurs canaux de communication. Cela permet d'une part d'identifier les acteurs de la communication, puis en interrompant la connexion, d'empêcher le contrôle des zombies par le maître, rendant le botnet inutilisable.

Afin d'évaluer les NIDS reposants sur cette dernière approche, il nous parait indispensable de modéliser le langage utilisé sur le *C&C* afin de générer un trafic qui puisse être à la fois contrôlable et réaliste.

Les travaux de Peter Wurzinger et al. [**?**] présentent une modélisation du comportement d'un zombie sur son canal de communication. Cependant, contrairement à nos travaux, le modèle utilisé ne prend pas en compte l'interdépendance des messages au sein d'une séquence de commandes. Cette modélisation à deux états (réception d'une commande, réponse à cette commande) engendre un trop fort risque de faux positifs.

Les travaux de Guofei Gu et al. [**?**] introduisent la corrélation spacio-temporelle des acteurs d'un *C&C* pour modéliser un botnet. Cependant, il apparait que cette solution est beaucoup trop tributaire de la synchronisation temporelle des acteurs d'un botnet. Si cette condition n'est pas assurée (QoS, mutation des codes, etc.), un même botnet ne sera pas modélisé de la même manière.

Chian Yuan Cho et al. [**?**] s'intéressent également à la modélisation des zombies. Leurs travaux proposent notamment l'utilisation d'une machine de Mealy pour modéliser le comportement d'un zombie sur son canal de communication. Ce modèle à deux limitations : d'une part, il ne prend pas en compte le temps séparant deux messages ; d'autre part, il suppose, de part son déterminisme, l'envoi d'une même réponse à un même message, quelle que soit l'instance du zombie. Cette modélisation est incompatible avec certaines commandes, par exemple, la commande *who*[4] sous Unix. Il est donc nécessaire d'intégrer un facteur d'indéterminisme sur les messages de sorties pour représenter ces différents cas. Pour se faire, nous utilisons un modèle indéterministe que nous détaillons au travers des contributions suivantes :

- Nous employons une machine de Mealy stochastique afin d'étendre la modélisation aux cas de botnets com-

[2] http://twitter.com

[3] http://www.shadowserver.org

[4] Who : commande transmettant la liste des utilisateurs connectés

plexes ainsi que pour accélérer l'apprentissage. Cette machine séquentielle stochastique permet notamment de fusionner les modèles associés à plusieurs instances d'un même zombie.
- Nous présentons un algorithme d'apprentissage d'une machine de Mealy stochastique en utilisant l'inférence passive et active de sa grammaire.
- Nous proposons une solution d'évaluation des mécanismes de détection d'un botnet en utilisant un générateur de trafic réaliste et contrôlable.

## III. Modélisation d'un zombie par MMSTD

Une des solutions pour évaluer un NIDS consiste à utiliser de vrai zombies. Cependant, les déployer n'est pas une opération triviale. En effet, même confiné sur un réseau protégé, le zombie risque de se comporter différemment s'il n'a pas l'autorisation de réaliser ses commandes malicieuses. En outre, certains zombies mettent en place des solutions de protection face aux tentatives de manipulation des experts (anti-virtualisation, obscurcissement, etc.) rendant leurs analyses longues et fastidieuses. Modéliser un zombie en vue de simuler son comportement permet donc de simplifier la génération de trafic.

Afin de limiter le nombre de paramètres décrivant le comportement du zombie, nous considérons que la modélisation des échanges sur un canal de communication se réduit à la modélisation de la grammaire employée par ses acteurs. En effet, nous ne considérons pas les paramètres d'une transmission tels que les erreurs, le bruit de fond et les délais de propagation spécifiques au support électronique. La modélisation du *C&C* d'un botnet nécessite donc la modélisation de ses acteurs : le maître et le zombie.

La modélisation d'un maître est une opération compliquée puisque qu'elle implique très souvent (dans le cas des architectures non P2P) l'analyse du comportement d'un humain. En outre, elle nécessite dans un premier temps de connaître le langage utilisé par le zombie. Nous avons donc choisi de nous intéresser à la modélisation du zombie et de considérer ultérieurement celle du maître.

Pour toutes ces raisons, nous avons fait le choix de modéliser le comportement d'un zombie sur son *C&C* pour pouvoir créer ensuite un zombie équivalent à partir de ce modèle. Ce dernier est ensuite utilisé pour la génération d'un trafic réaliste et contrôlable d'un botnet.

Parmi les modèles à base d'automates intégrant l'indéterminisme nécessaire, la famille des modèles de Markov cachés (HMM[5]) constitue un sous-ensemble aujourd'hui très utilisé. Un HMM [**?**] est une spécialisation des machines stochastiques où l'alphabet d'entrée est vide. Toutefois, nous considérons que ce modèle n'est pas adapté à notre problématique puisqu'il ne permet pas aisément de modéliser les communications d'un botnet. En effet, il est nécessaire de prendre en compte la relation entre les messages d'entrée (les commandes du maître) et les messages de sortie associés (les réponses du zombie).

[5]en anglais, Hidden Markov Models

Il existe également une extension des HMM dénommée IOHMM[6], où l'émission d'un message et la transition dépendent à la fois de l'état courant et de la réception d'un message, tout en étant probabiliste. Ce modèle caractérise la relation entre une séquence d'entrée et une séquence de sortie. Il est similaire à la machine de Mealy stochastique, qui est notamment évoquée par Paz [**?**] et qui nous semble particulièrement adaptée à notre problématique.

### *A. La machine de Mealy stochastique à transitions déterministes (MMSTD)*

La machine de Mealy stochastique (MMS), membre de la famille des machines séquentielles stochastiques (SSM[7]), est introduite par plusieurs travaux indépendants, publiés dès le début des années 1960. La modélisation suivante est tirée de Paz [**?**] :

*Définition 3.1:* Machine séquentielles stochastiques (SSM)
Une SSM est un quintuplet $M = \langle\ S, X, Y, T, q_0\ \rangle$ où $S$ représente l'ensemble des états, $X$ et $Y$, les alphabets d'entrée et de sortie et $q_0$ l'état initial de la machine. $T$ est un ensemble fini de matrices carrées, avec $|T| = |X| \times |Y|$, tel que $T = \{A(y|x)\}$ avec $y \in Y$ et $\in X$. Les matrices $A(y|x)$ définissent les probabilités de l'émission d'un message $y$ après réception d'un message $x$.

Dans notre cas, l'alphabet $X$ regroupe l'ensemble des messages envoyés par le maître au zombie et l'alphabet $Y$, l'ensemble des réponses envoyées par le zombie au maître.

La probabilité de l'émission d'un message $y$ suite à la réception d'un message $x$ est définie par la matrice de transition $A(y|x) = \{a_{i,j}(y|x)\}$. Chaque élément de cette matrice est de la forme $a_{i,j}(y|x) = p(s_j, y|s_i, x)$ et exprime la probabilité de passer à l'état $s_j$ en émettant le message $y$ suite à la réception du message $x$ alors que l'état courant est $s_i$.

Nous faisons l'hypothèse que les différentes instances d'un même zombie possèdent un graphe de transition commun, mais qu'elles peuvent émettre des messages différents. En effet, ces derniers caractérisent l'environnement au sein duquel chaque instance évolue. À partir de cette hypothèse, nous faisons le choix d'utiliser la version stochastique de la machine de Mealy à transitions déterministes et à sorties indéterministes. En outre, cette extension permet de factoriser la représentation du modèle qui caractérise différentes instances d'un même zombie, en offrant la possibilité d'associer plusieurs sorties au même couple (état, entrée).

Cette approche permet de modéliser l'ensemble des zombies à transitions déterministes et utilisant un vocabulaire invariant. Cela exclut donc les codes évolutifs ainsi que les botnets à communications chiffrées qui représentent une faible proportion des botnets existants.

*Définition 3.2:* Machine de Mealy stochastique à transitions déterministes (MMSTD)

[6]en anglais, Input/Output Hidden Markov Models.
[7]en anglais, Stochastic Sequential Machine.

Une MMSTD est une SSM telle que pour un état donné et un message reçu, il existe une seule transition possible :

$$\forall x \in X, \forall s_i, s_j \in S \quad p(s_j|s_i, x) \in \{0, 1\} \tag{1}$$

*B. La prise en compte du temps de réaction*

Comme présenté dans la section II, plusieurs mécanismes de détection de botnet utilisent la corrélation temporelle. Pour évaluer ces solutions, il est donc nécessaire d'intégrer un facteur temporel dans notre modèle, ce qui n'est pas le cas, par défaut, des modèles à bases de MMS.

Pour ce faire, nous ajoutons à la définition d'une transition, une variable aléatoire $T$ traduisant le temps de réaction avant l'émission d'une réponse. Cette variable suit une loi normale que nous paramétrons avec la moyenne $m$ et l'écart type $s$ des temps de réaction observés.

*C. La réduction de l'alphabet d'entrée*

Dans le cas général, nous ne distinguons pas les paramètres des commandes dans les messages échangés. Ainsi nous considérons « *.debug false* » et « *.debug true* » comme deux messages distincts.

Toutefois, cette approche complexifie le modèle. Cela est particulièrement vrai pour les commandes présentant les caractéristiques suivantes :

- des paramètres dont le domaine est de taille importante (par exemple, des adresses IP, des dates, etc.) ;
- une variation de ces paramètres n'a pas d'impact sur la séquence des actions réalisées mais peut conduire à des sorties différentes.

Nous avons donc décidé de distinguer certains paramètres que nous représentons sous la forme d'un motif qui abstrait les différentes valeurs qu'ils peuvent prendre. L'intégration de ces motifs, de la forme ${type_paramètre}, dans l'alphabet d'entrée permet de réduire sa taille et simplifie le modèle.

Pour un ensemble de messages d'entrée considérés comme équivalents, il devient possible de créer une seule transition comprenant l'ensemble de messages de sorties associé. Parmi les motifs utilisés, nous retrouvons :

- $IP, qui représente une adresse IP ;
- $EMAIL, qui représente une adresse email ;
- $DATE, qui représente une date au format yyyy-mm-dd ;
- $HOUR, qui représente une heure au format hh :mm :ss.

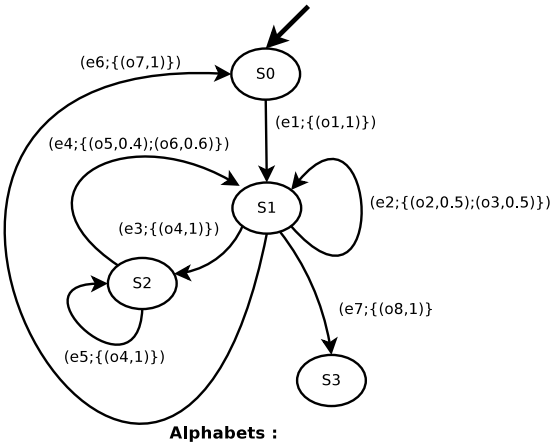
Chaque message $x \in X$ est donc de la forme : [<commande> | <motif>]* avec <commande> une chaîne de caractères non typée et <motif> un paramètre identifié par son motif.

La figure 1 illustre la modélisation d'un zombie par une machine de Mealy stochastique à transitions déterministes. Les transitions y sont représentées à l'aide d'une flèche orientée et utilisent la notation $(e, \{(o_i, p(o_i))\})$ avec $e$ le message d'entrée, $o_i$ un message de sortie et $p(o_i)$ la probabilité d'avoir $o_i$.

Des paramètres génériques (motifs) et l'opérateur de négation (!) sont utilisés dans l'alphabet d'entrée. Pour plus de lisibilité, la figure 1 n'intégre pas les termes relatifs aux temps de réaction. Ainsi, une fois dans l'état **S2** le zombie passe à l'état :

S1 s'il reçoit le message $e4$=$IP avec $IP le motif décrivant n'importe quelle adresse IP. Cette transition émet alors le message $o5$=« Attack successfull » avec une probabilité de $0,4$ et le message $o6$=« Attack failed » avec une probabilité de $0,6$ ;

S2 s'il reçoit le message $e5$= !$IP avec !$IP le motif décrivant n'importe quel contenu différent d'une adresse IP. Cette transition émet alors le message $o4$=« IP of target ? ».




**Alphabets :**

| *Entrées* | *Sorties* |
|---|---|
| e1=".login 123" | o1="Welcome master" |
| e2=".info" | o2="Windows XP" |
| e3=".ddos" | o3="Linux" |
| e4=$IP | o4="IP of target ?" |
| e5=!$IP | o5="Attack successfull" |
| e6=".logout" | o6="Attack failed" |
| e7=".disconnect" | o7="Goodbye" |
| | o8="Disconnected" |

FIGURE 1. Exemple d'une MMSTD d'un botnet

## IV. Apprentissage automatique de la MMSTD

Pour réaliser la simulation du *C&C* d'un botnet, nous utilisons le résultat de la modélisation d'un zombie. Cette modélisation permet d'obtenir un générateur de trafic non-malicieux mais disposant d'une grammaire équivalente à celle utilisée par des zombies. Toutefois, l'équivalence dépend des besoins de l'utilisation du modèle. Dans notre cas, celui-ci doit permettre d'évaluer une solution de détection des botnets. Le trafic généré doit donc respecter l'ensemble des caractéristiques exploitées par ces solutions de détection. Nous avons donc dressé la liste des équivalences importantes à considérer :

**Equivalence syntaxique :** les solutions de détection d'un *C&C* utilisent le vocabulaire du zombie ou du maître pour réaliser des règles de détection. Il est donc important d'apprendre le vocabulaire utilisé.

**Equivalence sémantique :** en plus de considérer le vocabulaire, certaines solutions de détection utilisent l'ordre d'apparition de ces messages dans la signature de détection d'un *C&C*. Il faut donc apprendre la grammaire, c'est-à-dire l'ensemble des états et des transitions du modèle du zombie, pour pouvoir simuler des enchaînements de messages réalistes.

**Equivalence temporelle :** finalement, une dernière caractéristique employée concerne la répartition dans le temps de l'échange de messages. Les solutions telles que [**?**] mesurent le temps entre deux messages pour déterminer si l'échange étudié est le résultat d'une communication d'un zombie et de son maître.

L'apprentissage du modèle se déroule en quatre étapes.

1) Dans un premier temps, nous effectuons une (ou plusieurs) capture(s) réseau des échanges entre un maître et une (ou plusieurs) instance(s) de zombies. Ces captures sont réalisées dans un environnement réaliste et non confiné.
2) Dans un second temps, nous analysons ces captures afin d'extraire l'alphabet d'entrée, le temps entre les messages ainsi que des séquences complètes d'utilisation de ces messages. Cette étape nous permet d'assurer l'équivalence syntaxique et temporelle de notre apprentissage.
3) Dans un troisième temps, nous téléchargeons et nous confinons le zombie au sein d'un environnement maîtrisé et invariant. Nous réalisons ensuite l'apprentissage du graphe de transition de cette instance du zombie, en simulant un maître. Cette étape garanti l'équivalence sémantique.
4) Finalement, la dernière étape consiste à généraliser les sorties du modèle en intégrant, dans le graphe de transition extrait à l'étape précédente, l'ensemble des séquences de messages observés dans les traces réseaux.

### *A. Capture des échanges entre une instance de zombie et son maître*

La première étape consiste à capturer des échanges entre une ou plusieurs instances d'un même zombie et un maître. Les traces doivent contenir l'enregistrement complet de chaque message échangé incluant son horodatage. Il existe différentes méthodes pour capturer ces messages : au niveau du réseau, au niveau du *C&C* ou au niveau de la mémoire, en fonction des besoins (chiffrement des communications).

### *B. Extraction des alphabets, du temps de réaction et des séquences valides*

À partir d'un ensemble de traces réseau, notre système détermine les alphabets d'entrée et de sortie, le temps entre chaque message ainsi qu'une ou plusieurs séquences de messages valides. Les couples $(x_i, y_i)$ associés à la capture d'une commande $x_i$ et à sa réponse $y_i$ sont extraits. Les alphabets d'entrés $X$ et de sorties $Y$ sont ensuite définis tels que $X = \{\epsilon, x_1, x_2, ...\}$ et $Y = \{\epsilon, y_1, y_2, ...\}$ avec $\epsilon$ un message vide.

En plus de l'alphabet, cette étape permet également de mesurer le temps entre la réception d'un message et la réponse associée (*cf.* section III-B). Ces mesures correspondent au délai entre l'observation d'un message en destination du zombie et la réponse associée. Pour assurer la précision de cette mesure du temps de réaction, il est important de positionner l'observateur à une distance « temporelle » négligeable du zombie, devant le temps de réponse de ce dernier. Si cette condition ne peut pas être assurée, l'implémentation doit permettre de prendre en compte le délai associé à l'écart entre l'observateur et le zombie.

### *C. Extraction du graphe de transition*

Pour la troisième étape, nous utilisons une solution d'inférence active. Cette étape s'articule autour de la stimulation d'une entité appelée oracle par notre modélisateur, pour en inférer son graphe de transition. Nous utilisons en tant qu'oracle une instance d'un zombie confiné, auquel le modélisateur soumet des requêtes d'appartenances. En réponse à ces requêtes, l'oracle fournit une information booléenne indiquant si la requête soumise est grammaticalement correcte. Toutefois, l'oracle étant déterministe (il s'agit d'une instance particulière du zombie), il permet seulement de découvrir le graphe de transition du zombie et non les différentes réponses possibles (pour un même couple entrée/état).

Le choix des requêtes d'appartenance s'effectue selon l'algorithme L* développé par D. Angluin [**?**] pour l'apprentissage des langages réguliers. Cet algorithme a ensuite été adapté à l'apprentissage des machines de Mealy par Shahbaz et al. [**?**]. Nous avons fait le choix d'utiliser cet algorithme en raison de son efficacité à inférer un graphe de transition déterministe en un temps polynomial.

Nous avons donc utilisé la bibliothèque LearnLib [**?**], maintenue par l'université technologique de Dortmund[8], pour réaliser l'inférence du graphe de transition du zombie.

### *D. Mise à jour du graphe de transition*

La dernière étape consiste à généraliser notre modèle en intégrant l'indéterminisme sur les messages de sortie. Pour ce faire, nous utilisons les séquences extraites des traces réseaux (*cf.* section IV-B). En considérant que ces séquences sont valides, nous les comparons avec notre modèle inféré. L'oracle et le zombie possédant un graphe de transition équivalent, il est possible de simuler les séquences extraites à travers le modèle de l'oracle afin d'identifier les différences provenant uniquement de l'indéterministe des sorties du zombie.

Pour effectuer cette opération, nous appliquons l'algorithme 1 présenté en pseudo-code où nous retrouvons $e \in E$ une séquence de commandes $(x,\ y)$ avec $x \in X$

[8]http://faelis.cs.uni-dortmund.de

un message d'entrée et $y \in Y$ un message de sortie. La transition d'un état $s_i$ à un état $s_j$, suite à la lecture d'un message $x$, est définie par la fonction $s_j = \sigma(s_i, x)$. Finalement, la table $(table(s, x, y))$ permet de stocker le nombre d'occurrences de l'émission d'un message $y$ dans l'état $s$ sachant que le message $x$ a été reçu.

```
1 foreach {e in E}
2     s2 := q0
3     foreach {(x,y) in e}
4         s1 := s2
5         s2 := σ(s1,x)
6         table(s1,x,y) := table(s1,x,y)+1
7     end(foreach)
8 end(foreach)
```

Listing 1. Calcul du poids de l'émission d'un message y sachant que l'on est dans l'état s1 et que l'on réceptionne le message x

La probabilité de l'émission d'un message $y$ sachant qu'un message $x$ a été reçu et que l'automate se trouve dans un état $s$ est ensuite calculée de la manière suivante :

$$p(y|s,x) = \frac{table(s,x,y)}{\sum_{z \in Y} table(s,x,z)} \quad (2)$$

## V. Implémentation et validation de la plateforme

### A. Implémentation

Dans cette section, nous présentons l'implémentation de la modélisation et de l'apprentissage d'un botnet pour l'évaluation, ainsi que son application à la simulation d'un équipement de sécurité.

La figure 2 illustre l'architecture générale de la plateforme utilisée pour valider le bon fonctionnement de notre approche. Celle-ci est notamment composée des éléments suivants :

**Extracteur passif :** cet élément effectue l'extraction des communications entre un zombie et un maître à partir d'un enregistrement réseau au format PCAP. Développé en Java, ce module s'appuie sur la librairie jpcap[9] pour l'analyse des paquets IP. À partir de ces derniers, il extrait l'ensemble des communications entre plusieurs instances d'un zombie et un maître. L'extraction est ensuite mise à la disposition des autres modules sous la forme d'un document XML représentant les échanges.

**Modélisateur :** ce module effectue l'apprentissage actif de la grammaire du zombie en utilisant les résultats de l'extracteur passif et une instance du zombie comme oracle. Il implémente la solution d'apprentissage présentée dans le chapitre IV-C en déterminant les expériences à exécuter pour découvrir le modèle à inférer. Ces expériences sont soumises à l'oracle après qu'il soit positionné dans le contexte souhaité. À la fin de l'inférence, le modélisateur publie ses résultats sous un format XML représentant le modèle appris.

**Générateur de zombies :** ce module permet de créer un zombie à partir de la description XML de son modèle comportemental. En suivant les transitions déclarées dans le modèle, ce module respecte la grammaire, le vocabulaire et les durées inférées préalablement. Lorsque, pour une entrée donnée, le zombie générique peut émettre plusieurs réponses possibles, il effectue un tirage aléatoire selon la loi normale.

**Générateur de maître :** ce module génère un zombie générique dans le rôle du maître du botnet. Toutefois, contrairement au « Générateur de zombies », il choisit aléatoirement, parmi l'ensemble de ses possibilités, la transition qu'il souhaite emprunter dans le modèle du zombie inféré. La caractérisation de ce choix se traduit par l'émission d'un message aux zombies auxquel il est connecté.

### B. Expérimentation

Pour valider notre plateforme, nous avons appliqué nos travaux à l'inférence et à la simulation d'un botnet très répandu, à savoir le PHP/Bot[10] (ou pBot). Ce code malveillant est installé par le propriétaire du botnet sur des serveurs web vulnérables aux injections de codes PHP. Nous l'avons utilisé en raison du grand nombre de serveurs infectés (la recherche « *"class pbot", filetype :txt* » sur le moteur de recherche Google fournit près de 600 références) et de la facilité d'accès à son code source, ce qui nous a permis de valider le modèle inféré par rétroconception.

Après avoir transformé l'une de nos machines en zombie, nous avons capturés l'ensemble des messages échangés avec son maître sur le C&C préalablement identifié. Nous avons ensuite appliqué la méthode d'apprentissage présentée dans la section IV en utilisant le zombie ainsi que les captures réseaux.

La MMSTD représentée sur la figure 3 correspond au comportement du zombie appris (pour des raisons de lisibilité nous avons limité le nombre d'états représentés sur le schéma et nous n'avons pas représenté le temps de réaction). Après l'analyse des résultats, nous avons identifié une spécificité de la version utilisée du pBot : elle n'autorise pas l'exécution successive de la même commande. Le résultat est fidèle à l'analyse réalisée par rétroconception du code source du PHP/Bot. En outre, nous avons simulé le comportement du zombie sur un serveur IRC local et vérifié expérimentalement la bonne détection du zombie simulé par la règle « *BLEEDING-EDGE TROJAN pBot (PHP bot) Commands* » du NIDS Snort[11]

## VI. Conclusion et perspectives

Nous avons proposé une solution pour l'évaluation des mécanismes implémentés au sein des NIDS pour la détection de botnets. Cette solution est basée sur la modélisation et l'apprentissage automatique de l'utilisation du *C&C* par un zombie pour ensuite le simuler au sein

[9] http://sourceforge.net/projects/jpcap/

[10] http://owned-nets.blogspot.com

[11] http://www.snort.org

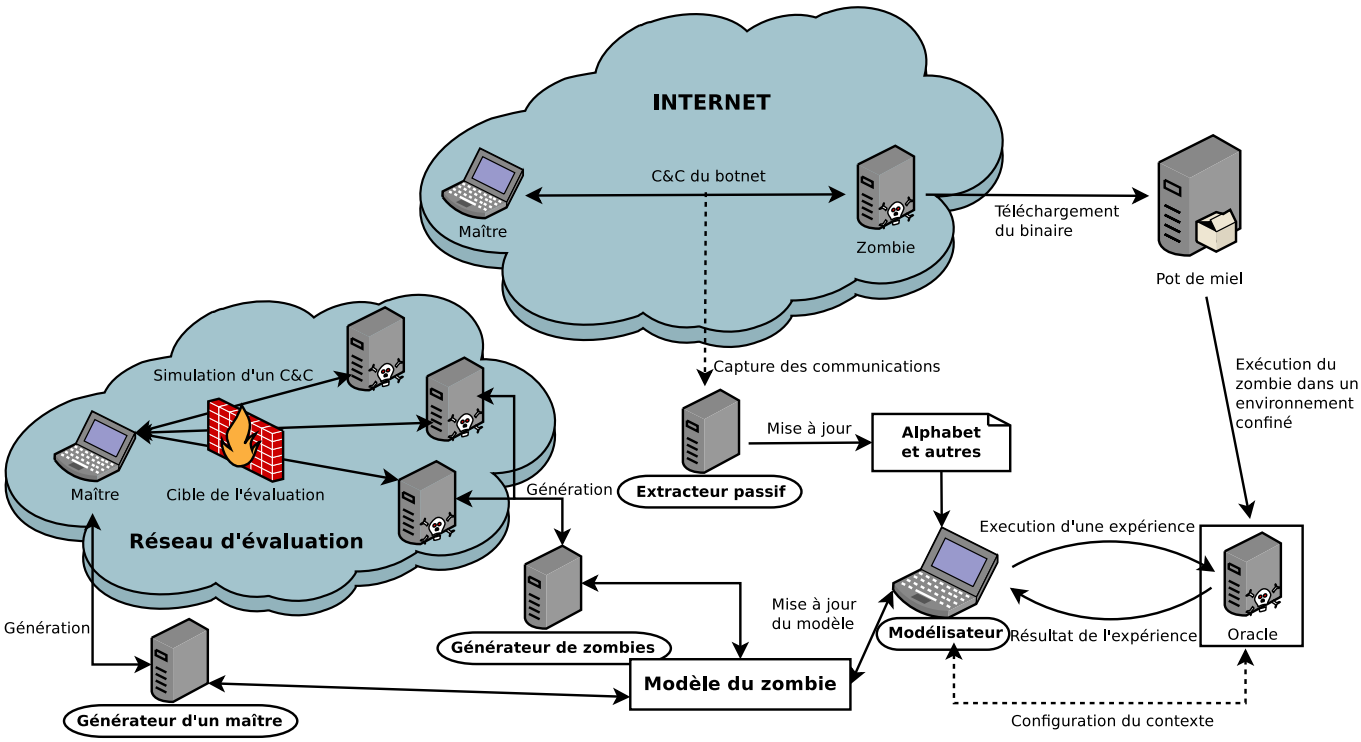



FIGURE 2. Architecture de la plateforme utilisée

d'une plateforme d'évaluation. Contrairement aux travaux antérieurs dans le domaine de la modélisation de zombies, nous utilisons une MMSTD afin d'étendre le comportement inféré aux cas indéterministes. En outre, nos travaux intègrent le temps de réaction d'un zombie et une factorisation du modèle afin d'assurer une plus grande équivalence entre le zombie inféré et le zombie simulé. Finalement, nous utilisons le modèle inféré pour générer un trafic réseau réaliste et contrôlable dans le cadre de l'évaluation des mécanismes de détection de botnets.

Dans un futur proche, nous allons automatiser la collecte de zombies en couplant notre infrastructure à celle d'un pot de miel dédié aux botnets. L'objectif est de pouvoir valider expérimentalement notre démarche sur un plus grand nombre de codes malveillants. Nous souhaitons également étendre notre approche en modélisant les actions malveillantes réalisées par un zombie suite à la réception d'une commande.

## RÉFÉRENCES


[1] D. Angluin. Learning regular sets from queries and counterexamples. *Inf. Comput.*, 75 :87–106, November 1987.

[2] P. Baecher, M. Koetter, M. Dornseif, and F. Freiling. The nepenthes platform : An efficient approach to collect malware. In *Proceedings of the 9 th International Symposium on Recent Advances in Intrusion Detection*, pages 165–184. Springer, 2006.

[3] J. R. Binkley. An algorithm for anomaly-based botnet detection. In *Proceedings of USENIX Steps to Reducing Unwanted Traffic on the Internet Workshop*, pages 43–48, 2006.

[4] M. Bishop, R. Crawford, B. Bhumiratana, L. Clark, and K. Levitt. Some problems in sanitizing network data. In *15th IEEE International Workshops on Enabling Technologies : Infrastructure for Collaborative Enterprises*, pages 307–312, 2006.

[5] J. Caballero, P. Poosankam, C. Kreibich, and D. Song. Dispatcher : enabling active botnet infiltration using automatic protocol reverse-engineering. In *Proceedings of the 16th ACM conference on Computer and communications security*, CCS '09, pages 621–634, New York, NY, USA, 2009. ACM.

[6] C. Y. Cho, D. Babić, E. C. R. Shin, and D. 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.

[7] H. Choi, H. Lee, and H. Kim. Botgad : detecting botnets by capturing group activities in network traffic. In *Proceedings of the Fourth International ICST Conference on COMmunication System softWAre and middlewaRE*, COMSWARE '09, pages 2 :1–2 :8, New York, NY, USA, 2009. ACM.

[8] D. Dittrich and S. Dietrich. P2p as botnet command and control : a deeper insight. In *Proceedings of the 3rd International Conference On Malicious and Unwanted Software*, pages 46–63, 2008.

[9] R. F. Erbacher, A. Cutler, P. Banerjee, and J. Marshall. A multi-layered approach to botnet detection. In *Security and Management*, pages 301–308, 2008.

[10] F. C. Freiling, T. Holz, and G. Wicherski. Botnet tracking : Exploring a root-cause methodology to prevent distributed denial-of-service attacks. In *Proceedings of 10 th European Symposium on Research in Computer Security*, pages 319–335, 2005.

[11] F. Giroire, J. Chandrashekar, N. Taft, E. Schooler, and D. Papagiannaki. Exploiting temporal persistence to detect covert botnet channels. In *Proceedings of the 12th International Symposium on Recent Advances in Intrusion Detection*, 2009.

[12] S. Gorbunov and A. Rosenbloom. Autofuzz : Automated network protocol fuzzing framework. In *IJCSNS International Journal of Computer Science and Network Security*, volume 10, August 2010.

[13] G. Gu, P. Porras, V. Yegneswaran, M. Fong, and W. Lee. Bothunter : detecting malware infection through ids-driven dialog correlation. In *Proceedings of 16th USENIX Security Symposium on USENIX Security Symposium*, pages 12 :1–12 :16,

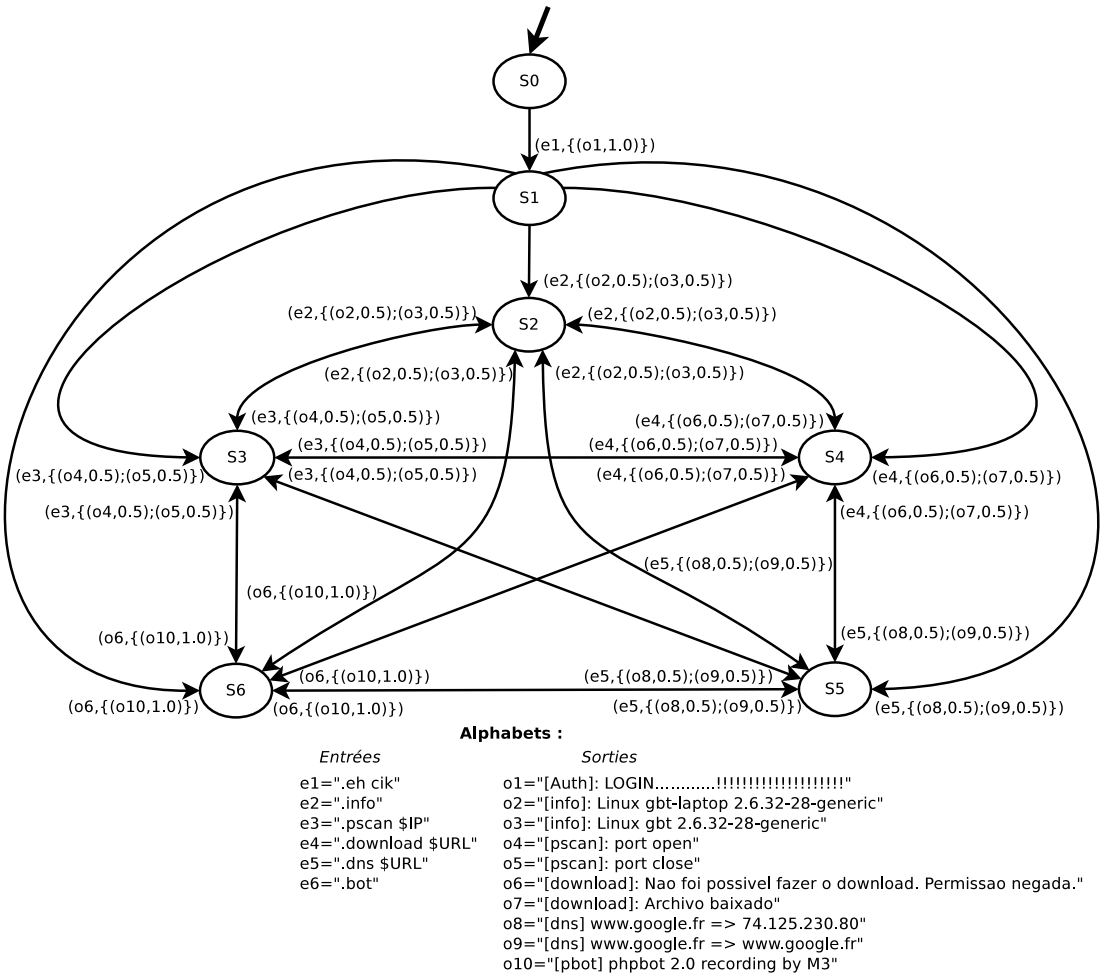



FIGURE 3. La MMSTD simplifiée du zombie PHP/Bot

Berkeley, CA, USA, 2007. USENIX Association.

[14] G. Gu, J. Zhang, and W. Lee. Botsniffer : Detecting botnet command and control channels in network traffic. In *NDSS*, 2008.

[15] S. Luo and G. A. Marin. Modeling networking protocols to test intrusion detection systems. In *Proceedings of the 29th Annual IEEE International Conference on Local Computer Networks*, LCN '04, pages 774–775, Washington, DC, USA, 2004. IEEE Computer Society.

[16] V. Nivargi, M. Bhaowal, and T. Lee. Machine learning based botnet detection. Technical report, CS229, Standford, 2006.

[17] A. Paz. *Introduction to Probabilistic Automata*. Academic Press, 1971.

[18] L. R. Rabiner. A tutorial on hidden markov models and selected applications in speech recognition. In *Proceedings of the IEEE*, pages 257–286, 1989.

[19] H. Raffelt, B. Steffen, and T. Berg. Learnlib : a library for automata learning and experimentation. In *Proceedings of the 10th international workshop on Formal methods for industrial critical systems*, FMICS '05, pages 62–71, New York, NY, USA, 2005. ACM.

[20] M. Shahbaz and R. Groz. Inferring mealy machines. In *Proceedings of the 2nd World Congress on Formal Methods*, FM '09, pages 207–222, Berlin, Heidelberg, 2009. Springer-Verlag.

[21] C. Wright, C. Connelly, T. Braje, J. Rabek, L. Rossey, and R. Cunningham. Generating client workloads and high-fidelity network traffic for controllable, repeatable experiments in computer security. In *Proceedings of the 13th International Symposium on Recent Advances in Intrusion Detection*, 2010.

[22] P. Wurzinger, L. Bilge, T. Holz, J. Goebel, C. Kruegel, and E. Kirda. Automatically generating models for botnet detection. In *Proceedings of the 14th European conference on Research in computer security*, ESORICS'09, pages 232–249, Berlin, Heidelberg, 2009. Springer-Verlag.