MINISTÉRIO DA DEFESA
EXÉRCITO BRASILEIRO
DEPARTAMENTO DE CIÊNCIA E TECNOLOGIA
INSTITUTO MILITAR DE ENGENHARIA
CURSO DE MESTRADO EM SISTEMAS E COMPUTAÇÃO

CARLA CHRYSTINA DE CASTRO PACHECO FERREIRA

MODELAGEM E SIMULAÇÃO PARA GERAÇÃO DE *TRACES* DE *BOT* NA ARQUITETURA CENTRALIZADA

Rio de Janeiro
2014

INSTITUTO MILITAR DE ENGENHARIA

CARLA CHRYSTINA DE CASTRO PACHECO FERREIRA

MODELAGEM E SIMULAÇÃO PARA GERAÇÃO DE *TRACES* DE *BOT* NA ARQUITETURA CENTRALIZADA

Dissertação de Mestrado apresentada ao Curso de Mestrado em Sistemas e Computação do Instituto Militar de Engenharia, como requisito parcial para obtenção do título de Mestre em Sistemas e Computação.

Orientadores: Prof. Ronaldo Moreira Salles - Ph.D.
Prof. Ricardo Choren Noya - D.Sc.

Rio de Janeiro
2014

INSTITUTO MILITAR DE ENGENHARIA
Praça General Tibúrcio, 80-Praia Vermelha
Rio de Janeiro-RJ CEP 22290-270

003.3 Ferreira, Carla Chrystina de Castro Pacheco
F383m Modelagem e Simulação para Geração de *Traces* de *Bot* na Arquitetura Centralizada/ Carla Chrystina de Castro Pacheco Ferreira; orientada por Ronaldo Moreira Salles e Ricardo Choren Noya. – Rio de Janeiro: Instituto Militar de Engenharia, 2014.
93 p.: il.

Dissertação (mestrado) – Instituto Militar de Engenharia – Rio de Janeiro, 2014.

1. Curso de Sistemas e computação – teses. 2. Redes de Computadores – Medidas de Segurança 3. Defesa Cibernética 4. Botnets 5. Modelagem – Computação 6. Simulação – Computação. I. Salles, Ronaldo Moreira II. Noya, Ricardo Choren III. Modelagem e Simulação para Geração de Traces de Bot na Arquitetura Centralizada. IV. Instituto Militar de Engenharia.

CDD 003.3

INSTITUTO MILITAR DE ENGENHARIA

CARLA CHRYSTINA DE CASTRO PACHECO FERREIRA

MODELAGEM E SIMULAÇÃO PARA GERAÇÃO DE *TRACES* DE *BOT* NA ARQUITETURA CENTRALIZADA

Dissertação de Mestrado apresentada ao Curso de Mestrado em Sistemas e Computação do Instituto Militar de Engenharia, como requisito parcial para obtenção do título de Mestre em Sistemas e Computação.

Orientadores: Prof. Ronaldo Moreira Salles - Ph.D.
Prof. Ricardo Choren Noya - D.Sc.

Aprovada em 04 de fevereiro de 2014 pela seguinte Banca Examinadora:

Prof. Ronaldo Moreira Salles - Ph.D. do IME - Presidente

Prof. Ricardo Choren Noya - D.Sc. do IME

Prof. Anderson Fernandes Pereira dos Santos - D.Sc. do IME

Prof. Sidney Cunha de Lucena - D.Sc. da UNIRIO

Rio de Janeiro
2014

Dedico este trabalho a Minha Família!

AGRADECIMENTOS

Primeiramente, agradeço ao Verdadeiro DEUS, o Racional Superior e à sua Luz da Divina Providência que em conjunto com a Mãe Natureza iluminaram meu caminho para que eu encontrasse essa Casa do Saber, conhecida como IME, para mim tão querida e que já representa o divisor de águas em minha vida.

A minha Família que constitui a base de minha formação e é o suporte para que todo esse trabalho fosse possível de ser realizado. Todos os Professores da faculdade que fiz que deram força para eu seguir o caminho da Pesquisa, o qual tanto amo e descobri nele minha real vocação. Todos os Amigos que me deram força nessa jornada do saber que conclui sua primeira etapa com esse trabalho.

Uma dedicação especial aos meus orientadores e Professores Salles e Choren. Guiaram meu caminho para que esse sonho se tornasse realidade. São exemplos de profissionais de primeiríssima linha de competência, ética, caráter, inteligência, humanidade e sabedoria. Suas luzes me iluminam para que eu possa trilhar meu caminho rumo à realeza do Estudo, da Pesquisa, do Saber e do Compartilhamento do Conhecimento.

Fizeram parte dessa jornada, todos os professores e funcionários do Seção de Engenharia de Computação (SE/8) do Instituto Militar de Engenharia.

Meu Muito Obrigada!!!

Carla Pacheco

"Persistir, para evoluir e se desenvolver Racionalmente. Sem persistência, nada feito.

Para se construir qualquer coisa, é preciso persistir até construir e terminar."

Mestre Manoel Jacintho Coelho

SUMÁRIO

LISTA DE ILUSTRAÇÕES

LISTA DE TABELAS

LISTA DE ABREVIATURAS

ABREVIATURAS

AES	-	*Advanced Encryption Standard*
C&C	-	*Command and Control*
DDoS	-	*Distributed Denial of Service*
DES	-	*Data Encryption Standard*
DGA	-	*Domain Generation Algorithm*
DNS	-	*Domain Name System*
HTTP	-	*Hypertext Transfer Protocol*
IDS	-	*Intrusion Detection System*
IP	-	*Internet Protocol*
IRC	-	*Internet Relay Chat*
NAT	-	*Network Address Translation*
P2P	-	*Peer-to-Peer*
RST	-	*Reset*
TCP	-	*Transmission Control Protocol*

RESUMO

O número crescente de roubo de informações e de ataques que paralisam sistemas e serviços compõe o cenário de ameaças representadas pelas *botnets*. O acesso público aos rastros de seus tráfegos na rede (*traces*) é difícil de ser obtido. Este trabalho propõe uma modelagem e simulação configurável de *traces* de *bot*, baseando-se nas fases do ciclo de vida do *bot* na área de identificação e prevenção de atividades maliciosas. Essas são as fases que geram menor volume de tráfego. *Traces* simulados de *bot* foram gerados e, através da modelagem em fases, foi possível identificar o comportamento do *bot* na rede. Experimentos foram realizados e seus resultados confirmaram a qualidade e corretude da abordagem proposta.

ABSTRACT

The growing number of sensitive information stealing and network attacks compound the botnets threat scenario. Public access to botnet traces is difficult to obtain. This work models and proposes a simulation tool to generate bot traces based on the life cycle of a bot. The proposed model is based on the identification and prevention of malicious activities phases of the bot life cycle, which are the most subtle to detect. Simulated bot traces were generated and trough the modelled phases, the network bot behaviour was identified. Experiments were carried out and results obtained confirmed the quality and correctness of the proposed approach.

1 INTRODUÇÃO

A crescente ocorrência de fraudes eletrônicas, ataques distribuídos de negação de serviço (DDoS), envios de *spam*[1] em massa, dentre outras atividades maliciosas são fatores que despertam a atenção de pesquisadores e entidades da área de Defesa Cibernética. Ações como estas são articuladas através do uso de *botnets* que são redes formadas por computadores infectados por *malwares*[2] que realizam funções predefinidas de maneira automatizada (TYAGI, 2011), (SILVA, 2012) e (COOKE, 2005). Estima-se que, aproximadamente, de 16-25% dos computadores conectados à Internet participam de *botnets* (SILVA, 2012), (STURGEON, 2007) e (SADHAN, 2009). Um dos objetivos da propagação dos *malwares* também visa o crescimento das *botnets*, infectando novas máquinas e as transformando em *bots*[3].

Diversos trabalhos foram desenvolvidos na área de detecção e mitigação de atividades maliciosas sobre o tema *botnets*. Essas técnicas abrangem as atividades do *bot* de ataque e propagação. Como existe uma quantidade significativa de trabalhos deste tipo desenvolvidos por instituições acadêmicas, governamentais e empresas privadas de Segurança da Informação, optou-se por focar este trabalho na área de identificação e prevenção de atividades maliciosas.

Assim que a máquina é infectada, ocorre sua comunicação com o centro de Comando e Controle (C&C) para atualizar a versão do *malware*. Essa comunicação deixa rastros de tráfego na rede (*traces*) que servem de insumos para ferramentas de análise e detecção de *botnets*. Seu tráfego está cada vez mais adaptado para evadir os sistemas de detecção.

Há carência de *traces* reais com atividades de *botnets* ("*Missing Data Problem in Cyber Security Research*" (ABT, 2013b)) com acesso público para fins de estudo na área. Isso dificulta o desenvolvimento e a validação de novas técnicas de detecção (SILVA, 2012) e (TYAGI, 2011). Os motivos recaem sobre a possibilitade da existência de informações pessoais e/ou sigilosas nesses *traces* (SILVA, 2012).

Os poucos *traces* disponibilizados apresentam problemas nos fatores qualidade, repre-

[1]Em (TYAGI, 2011) e (XIE, 2008), são e-mails utilizados para propagação de *malware* e outras atividades maliciosas

[2]Em (STONE-GROSS, 2009), são definidos como códigos maliciosos

[3]Máquinas infectadas ou zumbis, conforme definido em (TYAGI, 2011).

sentatividade e contemporaneidade. Em 2011 e 2012, aproximadamente 76% dos pesquisadores das linhas de Segurança Cibernética construíram manualmente os dados de tráfego para seus trabalhos (ABT, 2013b). Dos *traces* disponibilizados para pesquisa, a duração da atividade maliciosa é curta e sua coleta é pontual. A visão conjunta desta atividade em vários pontos da rede e com acesso público não foi encontrada. Isso constitui um grande problema na área de validação de ferramentas de identificação, detecção e mitigação de *botnets*.

Devido à falta de *traces* com acesso público para pesquisa e experimentação, foram feitas propostas de ferramentas de geração de tráfego sintético de *botnet*. Essas propostas esbarram em limitações técnicas como a construção de ambientes fidedignos à realidade, especialmente em tamanho e processamento. Outro empecilho aos estudos é o fato do *malware* detectar quando é executado em uma *sandbox*[4] (LEE, 2009), pois pode modificar o seu comportamento, o que interfere na análise do pesquisador (WILLEMS, 2007).

Diversos trabalhos na área de detecção, incluindo técnicas de identificação de padrões na comunicação e mitigação na área de *botnets*, foram desenvolvidos. Essas propostas necessitam ser validadas com *traces*, que atuam como uma referência em áreas como auditoria, perícia, métricas, testes de ferramentas que lidam com tráfego de rede, dentre outros.

Este trabalho realiza a modelagem do comportamento do *bot* na rede e, por consequência, a geração de *traces* simulados do *bot*. O contexto engloba o protocolo `HTTP` com o mecanismo *pull* de busca de comando pelo *bot* pertencente à arquitetura centralizada. A justificação desta escolha se embasa no fato desta arquitetura ainda ser amplamente utilizada (SILVA, 2012) e na singeleza dos fluxos de comandos que ocorrem somente entre C&C para *bot* e *bot* para C&C. Já na arquitetura `P2P`, além dos fluxos ocorrerem de um *bot* para outro, também há um segundo nível de comunicação entre os supernós (LU, 2011), o que aumenta a complexidade da modelagem da representação desses fluxos (TRENDMICRO, 2006).

O objetivo é apresentar uma modelagem genérica do comportamento do *bot* na arquitetura centralizada com o mecanismo *pull* na área de identificação e prevenção de atividades maliciosas com o uso do protocolo `HTTP`.

Ferramentas que produzem *traces* sintéticos de *bots* ou *botnets*, por sua vez, são escassas, quando acessíveis. Isso é devido à complexidade da análise que envolve sua construção

[4]Em (THOMPSON, 2005), é um ambiente computacional isolado utilizado para realização de testes.

ser considerável, assim como a modelagem do *bot* ou de seu *trace* não ser trivial, o que impacta no fator qualidade do tráfego gerado. Algumas questões sobre possibilidades nessa área são:

- As fases do *bot* podem ser identificadas através de seu *trace*?
- Os *traces* de *botnets* podem ser gerados refletindo as fases do *bot*?
- A qualidade dos *traces* gerados pode ser comprovada?

O diferencial deste trabalho que responde a estas questões é a separação das atividades do *bot* por fases. Elas estão refletidas no *trace* gerado e podem ser identificadas em *traces* reais. Outro diferencial é a forma como essas fases foram implementadas. Estão sendo levadas em consideração, apenas, as fases referentes à área de identificação e prevenção de atividades maliciosas, incluindo a manutenção da conectividade do *bot*. Essas fases são as que geram menor volume de tráfego na rede, comparadas às que realizam ataque e propagação.

A importância do volume de tráfego está diretamente ligada à sutileza na detecção do tráfego malicioso. Quanto menor o volume, mais sutil é o tráfego à detecção. Ferramentas da área de mitigação e defesa foram propostas e diversos trabalhos, desenvolvidos. Pertencem à esta área as atividades de ataque e propagação de *malware*, que são as que geram maior volume de tráfego. Apesar da área de identificação e prevenção de atividades maliciosas possuir um menor número de trabalhos que a área de mitigação e defesa, é fato que as fases da primeira área citada ocupam a maior parte do ciclo de vida do *bot*, um espaço de tempo maior (mais durável) que as fases da segunda área. Isso motivou o desenvolvimento deste trabalho.

Para atingir este objetivo, foi realizado um estudo a respeito das atividades que o *bot* realiza antes da execução de ataques, roubo de dados e propagação. O foco é a área de comunicação do *bot* e o C&C, seu comportamento observável na rede, que tem como consequência o *trace* gerado. O *bot* é a unidade mínima da *botnet*, por isso ele foi escolhido como o ponto de partida de análise. A qualidade do tráfego simulado é comprovada através de experimentos que comparam a sua fidedignidade a artigos bem descritivos e a *traces* disponibilizados para análise. Por fim, foi feito um experimento com um conjunto de *bots*. Sua operação foi verificada e bem-sucedida.

As seções deste trabalho estão organizadas da seguinte forma: a Seção 2, "Conceitos sobre *Botnets*", descreve os conceitos necessários para o entendimento do cenário deste

trabalho. A Seção 3, “Trabalhos Relacionados”, cita trabalhos sobre modelagem das fases do *bot* e alguns *frameworks* com foco em geração de *trace* de *bot*. Prosseguindo para a Seção 4, “Modelagem”, que descreve como foi elaborada a modelagem das fases do *bot* e, em seguida, a Seção 5, “Simulação”, que foca em como foi feita a implementação do simulador que gera o *trace* do *bot*. Após, são apresentados os cenários e seus respectivos resultados validados com histogramas, testes quantitativos e comparação do sequenciamento na Seção 6, “Experimentos e Resultados”. Observações sobre os pontos principais do trabalho e a abertura de espaço para trabalhos futuros são feitas na Seção 7, “Conclusão”.

2 CONCEITOS SOBRE *BOTNETS*

Em (TYAGI, 2011), define-se *bot* como um termo genérico derivado de "ro-Bot" que é utilizado para descrever um *script* ou um conjunto de *scripts* projetados para desempenhar funções predefinidas de maneira automatizada. Neste trabalho, o contexto de todas as ações realizadas pelo *bot* é de cunho malicioso.

Botnets são redes formadas por computadores infectados por *malwares* que realizam funções predefinidas de forma automatizada, fato que compõe uma das grandes ameaças na atualidade (SILVA, 2012), (TYAGI, 2011), (COOKE, 2005), (CHOI, 2009) e (CERON, 2010). Cibercrimes e ciberataques, que incluem roubo de dados, fraudes, DDoS (*Distributed Denial of Service*), envios de *spam*[5] em massa, com estimativa de até 80% do tráfego mundial de e-mails (SILVA, 2012), dentre outros (TYAGI, 2011), são preocupantes na área de Defesa Cibernética. Estima-se que aproximadamente de 16-25% dos computadores conectados à Internet participam de *botnets* (SILVA, 2012), (STURGEON, 2007) e (SADHAN, 2009).

Esses *malwares* são espalhados na Internet através de propagação própria e visam captar a maior quantidade de *bots* para a formação de uma *botnet*, cujo controlador[6] pode alugar por quantidade, tempo e serviço determinados (SILVA, 2012) e (GONCHAROV, 2012), além de poder efetuar o ataque com seu exército cibernético particular.

O número de *bots* pertencentes a uma única *botnet* pode alcançar a casa dos seis dígitos (FOSSI, 2011) e (BINSALLEEH, 2010). Essas máquinas zumbis são controladas remotamente pelo *botmaster* e realizam ações, predefinidas no *malware*, ao seu comando. Como uma *botnet* pode conter computadores de várias partes do mundo, as questões legais sobre as atividades ilícitas recaem na situação político-jurídica existente entre as nações, o que dificulta a desarticulação de uma *botnet*.

A comunicação dos *bots* com os C&Cs produz rastros de tráfego na rede. Esses *traces* são de vital importância para os estudos de comparação dos resultados de ferramentas que se propõem a gerar e detectar tráfego de *botnet*. O acesso público para obtenção de

[5]Em (TYAGI, 2011) e (XIE, 2008), são e-mails utilizados para propagação do *malware* e outras atividades maliciosas.

[6]Em (SILVA, 2012), também referenciado como atacante, *botmaster* ou *botherder* é quem possui o controle da *botnet*.

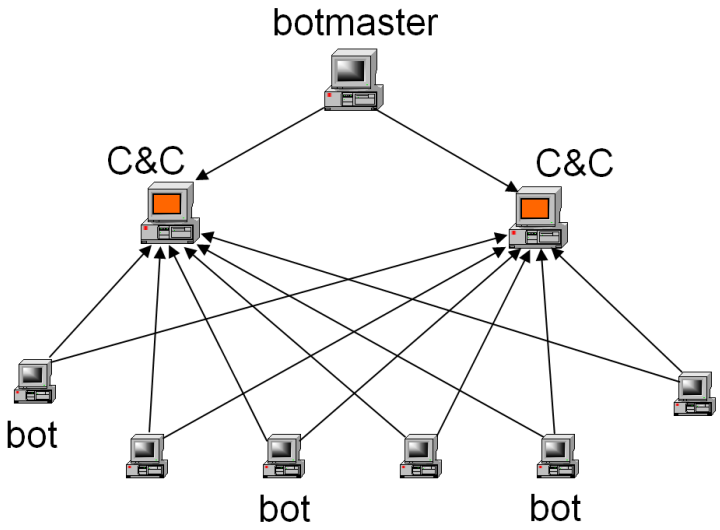


FIG. 2.1: Comunicação da *botnet* na arquitetura centralizada (WANG, 2007)

traces para estudo é difícil de ser obtido. As atividades de manutenção de conectividade do *bot* à *botnet*, assim como a atualização do *malware*, são as que geram menos volume de tráfego na rede se comparadas com as atividades de ataque e propagação. Portanto, neste trabalho serão abordadas as fases das atividades do *bot* que menos geram tráfego na rede.

Há um consenso na literatura produzida pela comunidade científica sobre a classificação das *botnets* no tocante ao tipo de arquitetura utilizada, pondendo ser **centralizada** (C&C) ou **distribuída** (P2P), conforme (SILVA, 2012), cap. 5 de (TYAGI, 2011), (LEE, 2009), (FEILY, 2009) e (YONG, 2012).

No tipo centralizada, o *botmaster* distribui o tráfego da *botnet* (comandos de ataque, atualizações e informações) em pontos chave denominados C&C que se comunicam diretamente com o *bot* (vide figura 2.1). Existem apenas os fluxos *bot* para C&C e C&C para *bot*. Já na arquitetura descentralizada, este fluxo pode ser distribuído entre os nós (*peers*) que podem apresentar o mesmo poder de distribuição de comandos.

A arquitetura descentralizada possui vários canais de comunicação de um *bot* para outro *bot* e, caso exista a figura do supernó (*bot* estratégico (STOVER, 2007)), de um *bot* para o C&C, como é o caso da *botnet* Storm. A latência na comunicação (alto tempo de resposta) entre os nós é maior do que nas *botnets* do tipo centralizada, portanto, é menor o sincronismo na comunicação na arquitetura descentralizada. O *botmaster* não possui rigidez no controle da *botnet*. O mecanismo *push* é mais utilizado neste tipo de arquite-

tura, porque os *bots* já estão conectados a um ou mais canais de comunicação. Então, o *botmaster* publica um arquivo com um *comando* ou envia um *comando* diretamente aos *bots* conectados a estes canais.

O trabalho irá se concentrar na arquitetura centralizada por sua simplicidade na modelagem e por ser largamente utilizada na atualidade (WANGA, 2010), (SILVA, 2012) e (FOSSI, 2011). Esta arquitetura permite ao *botmaster* um controle rígido da *botnet*, além da menor latência e maior sincronismo na comunicação, promovendo maior rapidez na disseminação de comandos. Quaisquer criptografias de canal de comunicação e autenticação do *bot* no C&C são desprezadas neste trabalho, visando manter a simplicidade do cenário abordado.

2.1 ARQUITETURA CENTRALIZADA (C&C)

Neste tipo de arquitetura, os *bots*, ao se conectarem à *botnet*, entram em contato com o C&C, que também atua como servidor de atualização. Tal fato é similar com o clássico modelo de rede cliente-servidor (SILVA, 2012). O *bot* recém infectado precisa possuir a lista destes servidores que é baixada juntamente com o *malware*. Esta lista atualizada pode estar em um arquivo de configuração inicial da *botnet* ou ser adquirida posteriormente em requisições ao C&C. Após essa conexão inicial, a atualização do *malware* é realizada.

O *bot* precisa saber a qual C&C ele deve se conectar, se unindo a um determinado canal IRC (GU, 2008a), no caso do utilização do protocolo IRC. Esses canais podem estar criptografados ou não (SILVA, 2012). Percebe-se que o C&C é um ponto importante para a desarticulação da *botnet*, pois centraliza o tráfego entre os *bots* e o *botmaster* (SILVA, 2012). Os *bots* fazem requisições a um servidor central, o que caracteriza a arquitetura centralizada (WANGA, 2010), conforme o ilustrado na figura 2.1. O *botmaster* controla diretamente a *botnet* (GU, 2008a) através dos C&C que possuem um canal de comunicação relativamente estável. Isso faz com que o C&C se torne um ponto fraco desta arquitetura (SILVA, 2012).

Nesta arquitetura, um *bot* não tem conhecimento sobre os outros *bots* da rede (STONE-GROSS, 2009). Apenas há uma hierarquia de comando dos servidores C&C para os *bots*. A comunicação entre o controlador e os *bots* inclui envio de comandos e recebimento de respostas em tempo real. Essas respostas podem ser entendidas como relatórios do resultado ou do progresso de uma tarefa executada pelo *bot* (GU, 2008a).

Existem redes baseadas no protocolo IRC. Porém, o tráfego IRC é facilmente bloqueado

por um *firewall* (SILVA, 2012), já que possui características bem peculiares. Por causa disso, o C&C baseado no protocolo HTTP possibilita o *botmaster* a se comunicar indiretamente com sua rede (GU, 2008a) e a manter a ofuscação da comunicação (SILVA, 2012). O motivo principal é que o tráfego *web* é permitido nas redes que fazem acesso a serviços *web* (STANKOVIC, 2009). Isso abrange as redes residenciais e corporativas. Com isso, os *bots* contactam os servidores C&C periodicamente em busca de comandos (GU, 2008a).

O tráfego de *botnets* possui baixo volume nas fases que não estão realizando ataque ou propagação do *malware*. Os *bots* ainda possuem comportamento espacial-temporal correlacionado (ex. janelas de tempo similares) de acordo com as suas atividades de respostas pré-programadas a comandos recebidos. Isso ocorre quando os *bots* possuem o mesmo perfil de atividade de tráfego ou enviam mensagens semelhantes. Esse comportamento se mantém por todo o ciclo de vida do *bot*. Vale destacar que, em um tráfego sem atividade maliciosa na rede, não ocorre o comportamento recorrente de respostas similares de clientes enviadas em uma janela de tempo semelhante (GU, 2008a) e (SADHAN, 2009). O alto sincronismo na comunicação com baixo tempo de resposta é um traço marcante desta arquitetura (SILVA, 2012), (GU, 2008a) e (SADHAN, 2009).

No mecanismo *push*, comandos são enviados pelos C&Cs aos *bots*. Já no mecanismo *pull*, os comandos são buscados pelos *bots*. No primeiro caso, os *bots* se conectam aos servidores C&C e aguardam o comando do *botmaster*. Quando um comando é enviado pelo atacante neste canal, todos os *bots* o recebem, o que garante ao *botmaster* o controle de sua *botnet* em tempo real. As *botnets* que utilizam o protocolo `IRC` são desse tipo. Já as baseadas no protocolo HTTP são do estilo *pull* (GU, 2008a).

Quando a *botnet* implementa o mecanismo *pull*, o *botmaster* injeta o comando num arquivo no servidor C&C. Os *bots*, então, periodicamente se conectam ao C&C para ler o comando neste arquivo. Para isso, os *bots* se conectam através de uma requisição a uma URL neste servidor e recebem o comando como resposta. Neste mecanismo, o *botmaster* não possui um controle em tempo real como no mecanismo *push* (GU, 2008a), mas ainda mantém o controle rígido da *botnet*. O mecanismo *pull* foi adotado para ser modelado e implementado neste trabalho por possuir menor volume de tráfego que o *push*.

A busca por comandos é sempre feita pelos *bots*. O perfil predominante de comunicação do *bot* é que segue o mecanismo *push* ou *pull*. Por causa de sua eficiência e eficácia, esta arquitetura deve se manter em uso futuramente (GU, 2008a).

3 TRABALHOS RELACIONADOS

Alguns trabalhos foram observados com foco na análise de modelos e ciclos de vida de *botnet*. Para tal, foram tomados como base artigos e outras produções científicas literárias da área de Segurança da Informação.

Em (SILVA, 2012), (TYAGI, 2011) e (FEILY, 2009), há pesquisas abrangentes feitas com a construção de uma espécie de catálogo das *botnets* ao longo do tempo. Conforme o observado nestes estudos, o foco da análise, especialmente na área de *análise da comunicação da rede* em termos de identificação de padrões de comunicação e comportamento na arquitetura cetralizada, norteou a construção deste trabalho.

3.1 MODELOS DE CICLO DE VIDA DA *BOTNET*

(XIN-LIANG, 2010) aborda propagação, tamanho, escala, estabilidade de *botnet* e seus modelos. Estes mesmos temas são abordados em (DAGON, 2006), (AJELLI, 2010) e (WANG, 2011a). Em (YONG, 2012), é feita uma proposta de um modelo matemático para a propagação e crescimento de uma *botnet*. Em (XIE, 2008), tem-se uma situação análoga a esta, pois trata de assinaturas e características de *botnets* do tipo *spammer*. (CERON, 2010) propõe uma metodologia de detecção baseado em padrões de assinaturas.

No trabalho da aplicação BotHunter (GU, 2007), são propostos estados genéricos de uma *botnet*, chamados de *Modelo de Diálogo de Infecção do Bot*, sendo:

- *Network scan*: escaneamento da rede para infecção de máquina vulnerável.
- *Victim exploit*: ações maliciosas na máquina recém infectada (o novo *bot*).
- *Binary download by the victim*: atualização do *malware* no *bot*.
- *Contact to a command and control*: estabelecimento de conexão com troca de mensagens com o C&C para manutenção da comunicação.
- *Outbound scanning*: escaneamento da rede a fim de propagar a infecção do *malware*.

Analogamente, a mesma estrutura é vista em (TYAGI, 2011) e (FEILY, 2009) na figura 3.1.

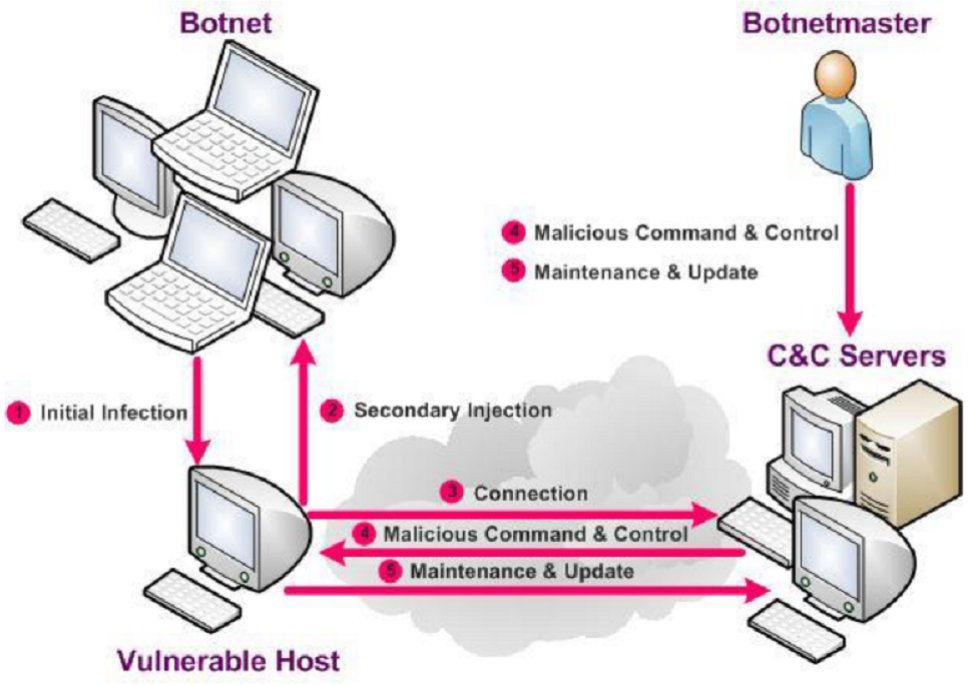


FIG. 3.1: Ciclo de vida de uma *botnet* (FEILY, 2009)

Em (SILVA, 2012) baseado em (FEILY, 2009) e (ZHU, 2008), é observada a seguinte estrutura:

- *Initial Infection*: infecção inicial.
- *Secondary injection*: procura e *download* de *malware*.
- *Connection or Rally*: estabelecimento de conexão com troca de mensagens com o C&C para manutenção da comunicação.
- *Malicious Activities*: realizar atividades maliciosas, o que inclui o ataque.
- *Maintenance and upgrading*: atualizações e troca de mensagens entre o *bot* e o C&C ou outros *peers*, no caso de uma arquitetura `P2P`, a fim do *bot* continuar participando da *botnet*.

Conforme o observado em (LILLARD, 2010), no capítulo 3 - *Other Network Evidence*, há uma coletânea sobre o ciclo de vida típico de uma *botnet*, o que passa pelos estágios já vistos com execuções de algumas atividades do *malware* na rede e no *bot*. BotGAD (*Botnet Group Activity Detector*) propõe uma metodologia para detecção de *botnet* baseado em observação do tráfego da rede (CHOI, 2009). Seus testes foram feitos em *traces* de sua própria instituição e a alta capacidade de processamento exigida se torna um complicador para a adoção desta metodologia em cenários reais com alto fluxo de tráfego na rede (SILVA, 2012).

(BABI, 2010) propõe uma máquina de estados de protocolo em redes de alta latência de *botnets* C&C. No estudo do levantamento da máquina de estados da *botnet* MegaD (*spamming*), foi verificada a dificuldade da determinação do número de estados, pois o algoritmo utilizado, baseado em autômatos determinísticos (houve dificuldade para a construção dos não determinísticos), mostrou que a quantidade de símbolos do alfabeto de entrada mantém ou muda o estado. No caso, a maior parte do alfabeto mantém o estado (*self-loops*), causando redundância no modelo. O tempo discretizado está incluso no alfabeto. Na comunicação, os palíndromos são considerados como sendo a mesma mensagem, o que contribui para a redução do número de estados.

Algumas limitações apresentadas são o grande número de consultas feitas ao *botmaster*, o que causa ruído nesta comunicação e provê a identificação de anormalidade de comportamento do *bot*, por conta do mapeamento de estados feito. Isso também pode afetar a construção da máquina de estados, já que o próprio *botmaster* pode rotear este *bot* para outro servidor interferindo na comunicação. A engenharia reversa do alfabeto inteiro é um problema a ser considerado.

Analogamente em (WANG, 2011b), há a consideração de cada mensagem levando a um estado, porém estas são distinguidas baseadas na distância de Jaccard. Após isso, uma classificação em um grupo é feita de acordo com a característica da mensagem extraída do tráfego e seus parâmetros previamente determinados. Então, um modelo probabilístico é inferido.

O BotSniffer foi construído com base no comportamento espacial-temporal observado da *botnet* em (GU, 2008a). Uma abordagem semelhante baseada em série temporal foi utilizada em (SADHAN, 2009). Estas propostas são da área de detecção e foram abordadas neste trabalho com o único propósito de citar os modelos existentes de ciclo de vida de *botnet*. Portanto, limitações destas propostas não serão feitas por isso estar fora do

escopo do presente trabalho.

Os modelos de ciclo de vida apresentados se referem a todas as fases de atividades do *bot*. O número de trabalhos nas áreas de ataque e propagação é bastante significativo. Essas são as fases que geram um maior volume de tráfego na rede.

Neste trabalho, somente as fases das atividades iniciais do ciclo de vida do *bot* e de manutenção de sua conectividade com a *botnet*, assim como a atualização do *malware*, são consideradas. Essas atividades geram menos volume de tráfego em relação às atividades de ataque e propagação. O comportamento temporal foi levado em consideração na modelagem das fases, como está descrito na Seção 4, "Modelagem".

3.2 *TRACES*

Diversos trabalhos na área de Segurança da Informação, incluindo técnicas de identificação de padrões na comunicação e mitigação na área de *botnets* foram desenvolvidos. Essas propostas necessitam ser validadas com *traces*, que atuam como uma referência em áreas como auditoria, perícia, métricas, testes de ferramentas que lidam com tráfego de rede, dentre outros.

Alguns estudos defendem a não desarticulação da *botnet*, como em (MCPHERSON, 2007), que apresenta um debate sobre a marcação do tráfego de *botnet* com sua devida invalidação (*blackhole*). Suas justificativas se respaldam na marcação e monitoramento do tráfego da *botnet*, a fim de se saber quando ele está ocorrendo na rede, servindo de alerta para os observadores, pois quaisquer mudanças em seu comportamento podem, assim, ser percebidas e estudadas para aprimoramento de técnicas de detecção. Além disso, é bastante útil para estudos de comparação de resultados de ferramentas que se propõem a gerar e detectar tráfego de *botnet* (*traces*[7]), já que a disponibilização do mesmo para análise é dificultada, pois pode conter informações pessoais e/ou sigilosas (SILVA, 2012). A comunicação de uma *botnet* é, geralmente, criptografada (LEE, 2009), (SILVA, 2012) e (TYAGI, 2011).

Lee (LEE, 2009) supõe o fato de um pesquisador unir-se voluntariamente a uma *botnet* para observar a comunicação entre o *bot* e a *botnet*. Isso implica diretamente em questões legais, pois o pesquisador estaria sendo negligente ao estar ciente de que seu computador pode vir a participar de um ataque real (ZOU, 2006). Outro senão seria o fato de sua

[7]Rastros deixados na rede contendo atividades dos pacotes pertencentes ao tráfego da *botnet* (SILVA, 2012)

máquina ser identificada pelo *botmaster* e ter o seu tráfego desviado, o que impactaria diretamente na coleta de evidência, comprometendo a pesquisa, além do fato da máquina do pesquisador estar exposta a atividades maliciosas que o *botmaster* pode vir a fazer (LEE, 2009).

Uma referência mundial de *traces*, porém antiga (1999-2000), é o DARPA (*Defense Advanced Research Projects Agency*). A atualização deste banco não acompanhou a evolução das técnicas e características das *botnets*. Ele possui *traces* de atividades maliciosas na área de ataque, que não é foco deste trabalho.

Devido a esta falta de material para estudo pelos pesquisadores, foram feitas propostas de ferramentas de geração de tráfego sintético de *botnet* (LEE, 2009). Dificuldades são encontradas pelos pesquisadores da área de engenharia reversa do *malware*, devido a constante mutação/adaptação e ofuscação de código (SILVA, 2012) e (LEE, 2009).

As propostas de ferramentas de geração de *traces* sintéticos de *botnets* esbarram em limitações técnicas, como a construção de ambientes fidedignos à realidade, especialmente em tamanho e processamento, com limitações de recursos computacionais nestes ambientes.

3.3 GERAÇÃO DE *TRACES* DE *BOT*

A seguir, serão citadas duas ferramentas que geram *traces* de *bot*. As limitações técnicas de ambas serviram de motivador para a construção do simulador proposto neste trabalho.

Christopher P. Lee, em sua tese de doutorado, construiu um *Framework for botnet emulation and analysis* (*Framework* para emulação e análise de *botnet*): o Rubot (LEE, 2009). Um problema é o fato da sua construção basear-se em funcionalidades advindas das *botnets* Storm e Nugache, ambas `P2P`. Observa-se que para a construção de métodos mais genéricos, abrangendo, de fato, outras *botnets* do tipo centralizadas, seriam necessários outros modelos de *botnets*, de preferência os mais representativos de ambas as arquiteturas. Além disso, o próprio autor relata algumas limitações de seu *framework*.

O Rubot, portanto, apresenta uma implementação não tanto genérica em termos de características de *botnets* em geral e, apesar de sua contribuição com geração de *traces* para o ramo científico, algumas funcionalidades necessitam ser desenvolvidas. Características relevantes a serem implementadas seriam as de suporte a testes, integrações do simulador, modelos de mensagem instantânea e de sensores, conforme o destacado:

> "Because botmasters attempt to evade detection and monitoring, it is often quite difficult for researchers to obtain visibility into the operation of the botnet. So not only is it sometimes hard to judge what the botnet is doing, its impossible to reproduce the actions. This does not lend itself to rich research. With the Rubot framework, results are reproducible and other researchers can verify the results and implementation. [...] However, the following four major features that warrant elevated attention: testbed support, simulator integration, instant messenger models, and sensor models." (LEE, 2009, p. 77; 80).

Outro *framework* para os mesmos fins é o SLINGbot (JACKSON, 2009) que foi construído baseado na taxonomia de um relatório técnico da TrendMicro de 2006 (TRENDMICRO, 2006). Possui um ambiente totalmente controlado de simulação de *botnet* e, em seus exemplos de *botnets* implementadas, há foco na utilização do mecanismo *push* em que a iniciativa da comunicação parte do C&C para o *bot*. Este mecanismo produz maior volume de tráfego que o *pull*, em que a iniciativa parte do *bot* para o C&C. O mecanismo *pull* é o mais utilizado nas comunicações da arquitetura centralizada (ZEIDANLOO, 2009).

O acesso ao *framework* SLINGbot não é público e não foi permitido quando sua equipe foi contactada. Já o Rubot permite o acesso para fins de estudos.

Somando-se a esta lista, em (ABT, 2013a), existe uma iniciativa em aberto de Sebastian Abt para a construção de um *framework* para simulação de *botnet* com o ambiente totalmente desenvolvido no simulador de rede `NS-3`[8].

Algumas dificuldades, além da carência de *traces* disponíveis para estudo, foram encontradas durante a pesquisa de artigos e trabalhos relacionados para a modelagem do simulador. O estudo deparou com a carência de artigos com citação do sequenciamento e exemplos de mensagens das transições do escopo deste trabalho e ainda com a falta de descrição minuciosa do comportamento na comunicação, na ocorrência de acertos e de erros, ou da inexistência de arquivos nos servidores.

Outras dificuldades em relação à implementação do simulador foram enfrentadas, como a comparação e marcação dos *traces*. A comparação foi feita através da análise dos arquivos no formato *dump* de rede, por não se tratar de grande volume de dados. Já a marcação foi resolvida com a adição do campo "`Marca-do-bot`" no cabeçalho `HTTP`.

[8] *The ns-3 network simulator* em http://www.nsnam.org/

Na tabela 3.1, está destacada a comparação das características das ferramentas SLINGbot e Rubot.

TAB. 3.1: Comparativo de características entre as ferramentas que geram *traces* de *bot*

Característica	**SLINGbot (JACKSON, 2009)**	**Rubot (LEE, 2009)**
Protocolo	Vários	Vários
Arquitetura & Ambiente	ambas (enfoque *push*) engloba *bot* e servidor ambiente controlado	ambas (enfoque `P2P`) engloba *bot* e servidor ambiente controlado
Acesso	restrito	público
Implementação	Python, conjunto de funcionalidades	Ruby, conjunto de funcionalidades

4 MODELAGEM

As fases modeladas do *bot* refletem o seu comportamento na rede, de acordo com critérios de sequenciamento, repetição, ocorrência de fluxo de comunicação e temporização. Artigos e relatórios técnicos da área de Segurança da Informação constituíram a maioria dos insumos para análise das fases do *bot* na área de identificação e prevenção de atividades maliciosas. Nesta seção, estão descritas as fases e como são feitas suas simulações. A validação dessa modelagem está representada pelos resultados dos experimentos simulados, conforme Seção 6, "Experimentos e Resultados".

A modelagem do comportamento geral do *bot* foi elaborada para se fazer o desenvolvimento do simulador. O foco foram as características gerais do *bot* na totalidade. Neste trabalho, é proposta a modelagem das fases do *bot* na arquitetura centralizada na área de identificação e prevenção de atividades maliciosas.

Foram adotados padrões para esta modelagem. Um deles é o protocolo `HTTP` que possui sua principal vantagem na camuflagem do tráfego de *botnet* com tráfego *web* (SILVA, 2012), (GU, 2008a), (WANG, 2010), o que facilita sua passagem por *firewalls* e evita sua detecção por IDS (ZEIDANLOO, 2010). Já o mecanismo *pull* foi adotado por ser o mais utilizado na arquitetura centralizada que faz uso do protocolo `HTTP`. A busca por comando parte do *bot* para o C&C (GU, 2008a), o que leva este mecanismo a produzir menos volume de tráfego que o *push*. As conexões com o C&C foram assumidas como aperiódicas (não persistentes) (SILVA, 2012) por refletir melhor o modelo `HTTP` utilizado. Essas conexões partem do *bot* para o C&C periodicamente, mas não são mantidas como já ocorre no caso dos canais de comunicação com o protocolo IRC em que o tempo médio de uma conexão pode ser de 25 minutos ou de 3 horas e 30 minutos.

Primeiramente, foi construída uma máquina de estados com todas as fases do ciclo de vida do *bot*, conforme o ilustrado na figura 4.1. Essa modelagem inicial foi feita a partir da literatura da área de *botnets* levando em consideração ambas as arquiteturas: centralizada e descentralizada. As mesmas nove fases (ou estados) existem em cada arquitetura e serão descritas sucintamente, a seguir:

- *Recém-Infecção*: abertura de conexões a um ou mais C&Cs, logo após a infecção da máquina.

- *Configuração*: requisições ao C&C de arquivos diferentes do executável do *malware*, como o arquivo de configuração do *bot* (MATROSOV, 2011).

- *Identificação*: requisição ao C&C com o único objetivo do envio do `IP` externo do *bot* à rede que ele pertence. Isso ocorre quando o *bot* está em rede protegida por *firewall* ou em rede com roteador, possuindo um `IP` local, atrás do `NAT`, o qual outros *bots* não conseguirão estabelecer conexão. Esta fase é opcional. Ela ocorre no caso do *botmaster* envolver o *bot* na gerência da *botnet* (BINSALLEEH, 2010). Por isso, é importante o *bot* saber o seu `IP` externo. Esta fase está ligada à fase "*Mudança de papel do bot na rede*".

- *Atualização*: requisição ao C&C para atualização do *malware*. Esta é a primeira fase iterativa. Deste estado, pode-se alcançar os estados de "Ataque", "*Report* do *status* do *bot*", "Verificação de conectividade à rede (*Keep alive*)", "Autopropagação" e "Mudança de papel do *bot* na rede". Percebe-se, no modelo, que este é um estado central que o *bot* necessariamente deve passar para poder alcançar um destes outros estados.

- *Ataque*: Execução do objetivo da *botnet*, como, por exemplo, roubo de senhas e credenciais, como é o caso da *botnet* Zeus (RICCARDI, 2013), ou envio volumoso de *spams*, como é o caso da *botnet* Storm (STOVER, 2007). *DDoS* também está contemplado nesta fase que representa o ataque da *botnet* em si.

- *Report do status do bot*: Controle do que ocorre em termos de tráfego local do *bot* e/ou respostas de sucesso sobre atualizações do *malware*.

- *Verificação de conectividade à rede (Keep-Alive)*: requisição enviada de tempos em tempos e quando a máquina é inicializada e se conecta à rede. O intuito é o *bot* comunicar à *botnet* que ele ainda pertence a ela. A periodicidade dos fluxos de comunicação neste estado são um traço marcante do *keep-alive*.

- *Autopropagação*: Também visa a sobrevivência da *botnet*. O *malware* se propaga através de ações de infecções, do envio de *spams* (Storm) ou de injeções de código em *sites* (Zeus). Varreduras na rede local e contaminação através de mídias removíveis também representam esta fase.

- *Mudança de papel do bot na rede*: Desinfecção da máquina ou mudança na característica de tráfego gerado na rede (HARLEY, 2010). O segundo caso pode ocorrer quando o *bot* apresenta um grande tamanho de banda de conexão e/ou quando sua localidade é estratégica para a *botnet*. Ocorrendo essa mudança de papel do *bot* na *botnet*, o *bot* passa a ser um publicador/replicador de comandos na rede (Storm), pode guardar informações da *botnet* ou ainda injetar o *malware* em diversos sites ou gerenciar a *botnet* (Zeus) (BINSALLEEH, 2010).

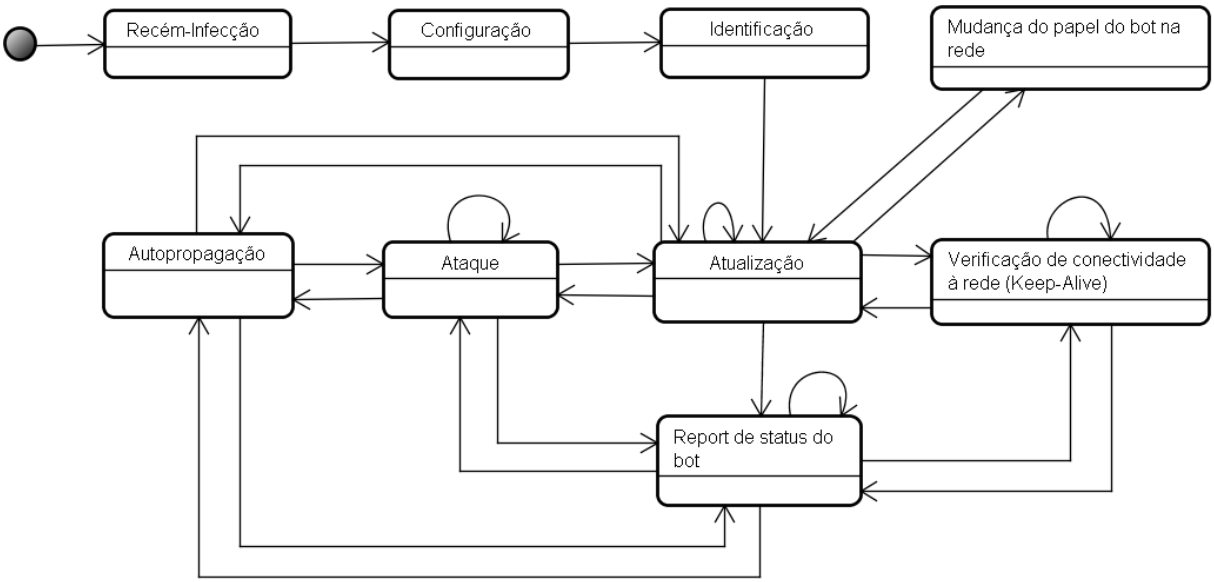


FIG. 4.1: Máquina de Estados Completa do *Bot*

Por este trabalho ser referente apenas às fases relacionadas à identificação e à prevenção de atividades maliciosas, apenas quatro fases foram destacadas para serem modeladas: Recém-Infecção, Configuração, Atualização e Keep-Alive. Então, uma segunda máquina de estados, com apenas essas quatro fases, foi construída e está ilustrada na figura 4.2. O refinamento deste modelo foi feito em um diagrama de atividades do *bot* na figura 4.3.

Atividades que ocorrem no estado "*Identificação*" estão relacionadas a atividades que ocorrem no estado "*Mudança do papel do bot na rede*", quando o padrão de comunicação muda por o *bot* realizar atividades ligadas à autopropagação e à replicação de comandos, o que nem sempre ocorre em todos os *bots*. Essas atividades produzem maior volume de trágfego e pertencem à área de detecção e defesa, assim como "*Ataque*" e "*Report do status do bot*", por isso, ficaram de fora da modelagem deste trabalho, mas com espaço aberto

para trabalhos futuros. Além disso, o foco foi a modelagem das atividades mais comuns à maioria dos *bots* para a composição de um modelo mais genérico.

4.1 FASES DO MODELO DE CICLO DE VIDA DO *BOT*

A fase Recém-Infecção foi identificada, porém não modelada por pertencer mais ao grupo de protocolos `P2P` e arquitetura descentralizada. Ela está presente nos diagramas para representar sua ocorrência em termos de sequenciamento das fases, já que é a fase inicial dos diagramas de estado e de atividades. Além disso, optou-se por manter o cenário simples com apenas um *bot* e um servidor. A abertura de múltiplas conexões que ocorre na fase Recém-Infecção foge deste cenário.

A simplificação do cenário visou o funcionamento da comunicação do *bot* para o servidor e do servidor para o *bot*. Esta comunicação funcionando corretamente confirma a corretude do simulador baseado no modelo. Com isso, cenários mais complexos com mais servidores podem ser incluídos, seja nos arquivos que contém a tabela de servidores para cada fase no simulador, seja com a extensão do simulador com o uso de algoritmos que geram nomes de domínios. O fato é que o funcionamento correto de um *bot* para um servidor facilita escalar a solução para mais servidores, pois o foco está no fluxo comunicação entre essas partes.

Ao todo, quantro fases foram identificadas e, apenas, três modeladas na área de identificação e prevenção de atividades maliciosas, conforme o mostrado pela máquina de estados na figura 4.2. Elas foram implementadas no simulador e ocorrem sequencialmente entre si, na ordem: Configuração, Atualização e Keep-Alive. A primeira e a última são opcionais, enquanto que a segunda deve sempre estar presente. Essas três fases, principalmente a Atualização e a Keep-Alive, são as que se mantém por mais tempo no ciclo de vida do *bot*. Por esses duas perdurarem no ciclo de vida do *bot*, utilizou-se um contêiner para destacar a iteração que ocorre em cada fase e entre as fases. Essas iterações são controladas por temporizadores.

As temporizações das fases Atualização e Keep-Alive são definidas no arquivo de configuração do simulador, assim como outros parâmetros. Essas duas fases são iterativas, enquanto que a de Configuração ocorre uma única vez. Um estudo sobre o comportamento temporal na rede é apresentado em (STRAYER, 2008) e (SILVA, 2013). Os temporizadores representam características dinâmicas que podem ser expressas em séries temporais. Essa característica é relevante para o simulador devido à geração do *trace* ser

contínua, conforme a execução das atividades do *bot* num cenário real.

A fase Atualização é a única obrigatória, pois o executável do *bot* precisa ser atualizado para ele se conectar a outros C&Cs ou executar novas técnicas de evasão, dentre outras ações (SILVA, 2012). Já a fase Configuração pode não existir em determinados *bots*, por isso é opcional. Como também é o caso da fase Keep-Alive, em que o comportamento de *keep-alive* pode estar incluso na fase Atualização, como é o caso do *bot* Confiker que está discutido na Seção 4.1.3, "Atualização".

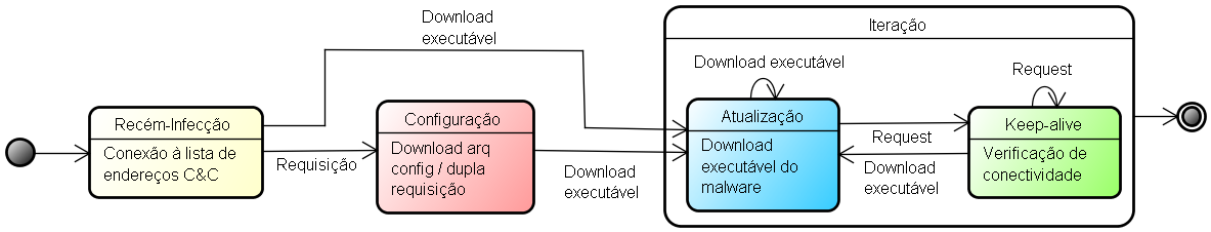


FIG. 4.2: Máquina de Estados do *Bot* das Fases identificadas na área de Identificação e Prevenção de atividades maliciosas

Adicionalmente, a análise deste diagrama (figura 4.2) foi aprofundada, resultando no diagrama das atividades do *bot*, mostrado na figura 4.3. As atividades ocorrem por fases e representam ações que ocorrem da mesma forma ou no mesmo momento. A heurística (e inteligência) do funcionamento geral das fases está implementada e é alimentada pelos parâmetros do arquivo de configuração do simulador que diferenciam os *bots*.

Está implementado apenas o cenário de sucesso, visando manter a simplicidade. Casos específicos de exceção na conexão, erro da resolução de nome para `IP`, inexistência de arquivo no C&C e insucesso na transferência de arquivo para o *bot* estão previstos na modelagem como atividades de exceção e seus tratamentos podem ser implementados. Analogamente, particularidades em montagens de cabeçalhos do protocolo `HTTP`, montagem e criptografia do *payload* nas requisições, tipos de envio de mensagens de sucesso, escolha e formação do nome dos servidores C&C, tratamentos de respostas do C&C (e interpretação de comando) em todas as fases, persistência na conexão e consulta a servidores C&C ou a *sites* podem ser implementadas para contemplar diferentes tipos de *botnets*.

O tratamento da interpretação de comandos leva em consideração o conteúdo da mensagem recebida pelo *bot*. Após o processamento da resposta da requisição, é invocado um um método abstrato para o tratamento de comandos. Em todas as fases, este método se encontra no fluxo de sucesso e pode ser estendido. Nenhum deles foi implementado, mas

o poderá ser feito em trabalhos futuros. Somente após o tratamento de comando é que a mensagem de sucesso é enviada nas fases Configuração e Atualização (figura 4.3).

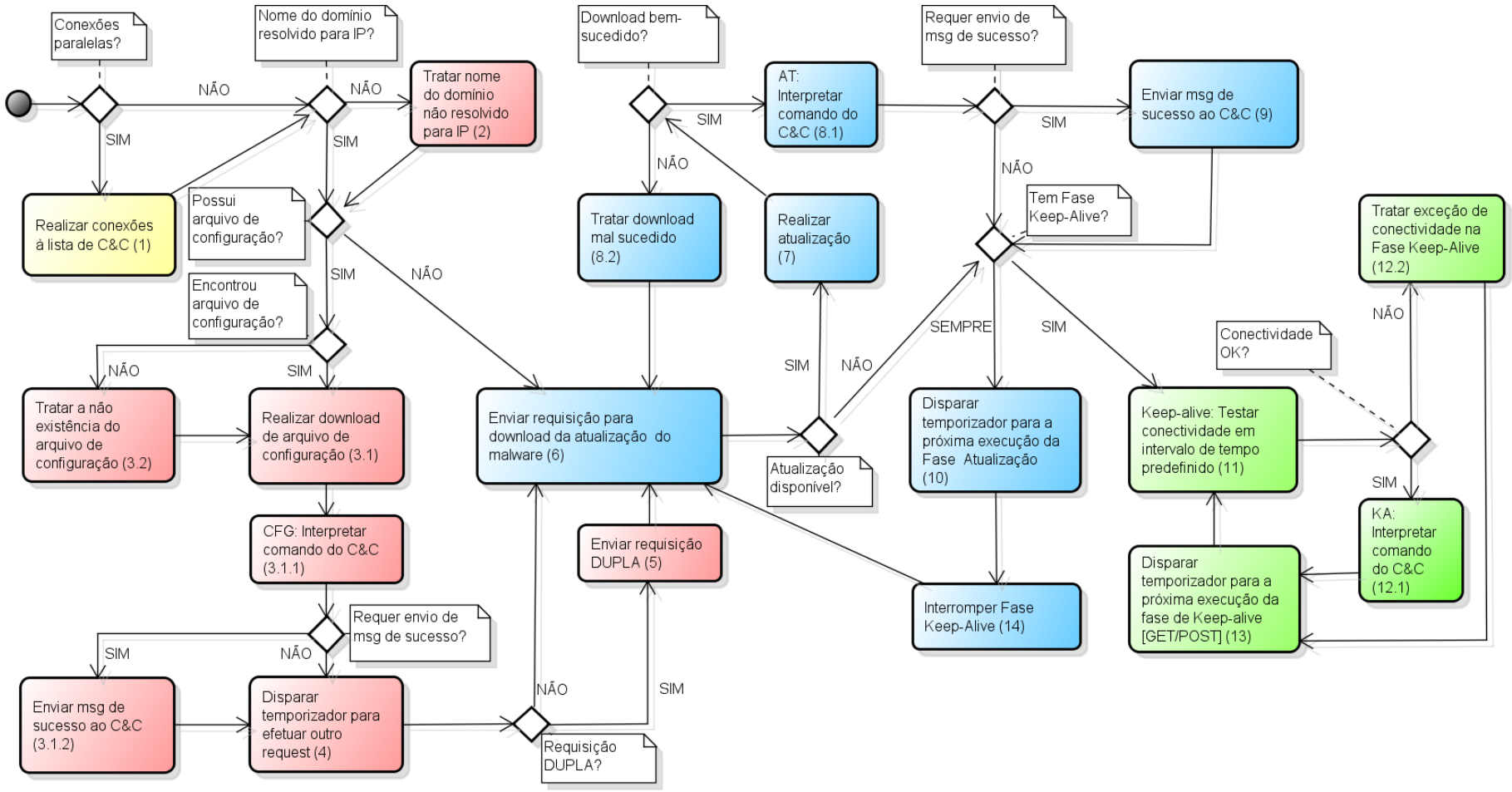


FIG. 4.3: Diagrama de Atividades das Fases identificadas e modeladas na área de Identificação e Prevenção de atividades maliciosas - Recém-Infecção: amarelo ; Configuração: rosa ; Atualização: azul ; Keep-Alive: verde

Cada atividade representa uma ou mais mensagens que chegam e/ou são enviadas pelo *bot* ou, ainda, ações que ele executa que levam à próxima atividade no diagrama. Portanto, a mudança de atividade ocorre com um evento que pode ser gerado pela chegada de outras mensagens ou pela passagem do tempo.

A representação de final do fluxo no diagrama de atividades (figura 4.3) foi determinada como sendo o caso de uma desinfecção da máquina. Isso pode ocorrer a qualquer momento do ciclo de vida do *bot*, portanto, optou-se por não representar um fluxo final neste diagrama.

Estão sendo representados os fluxos básicos e alguns fluxos opcionais de mensagens do *bot*. Como a ideia do simulador de *traces* de *bot* contempla a extensividade, optou-se por modelar também alguns fluxos opcionais a fim de citar o seu uso na implementação (ponto extensível do simulador).

Entende-se por básico, o fluxo que ocorre na maioria dos *bots* de várias *botnets*. Este fluxo contém as informações mínimas necessárias para a caracterização de um tráfego de *bot*. Para chegar à conclusão de que um fluxo é, ou não, básico, foram levadas em consideração as características e os comportamentos comumente encontrados nas mensagens dos *bots*. Um exemplo é a lista de servidores C&C com os quais o *bot* se comunica. Essa lista pode ser obtida com o *malware*, gerada automaticamente por um algoritmo ou ainda atualizada em uma determinada conexão do *bot* com o C&C. O fato é que a lista desses servidores C&C deve existir para que o *bot* se comunique com o C&C. Portanto, essa é uma característica classificada como básica e os fluxos que envolvem sua obtenção ou atualização também são classificados como básicos.

Outros fluxos básicos englobam a temporização das fases, a atualização do executável do *malware*, o sequenciamento das fases, suas repetições, enfim, o que está envolvido no cenário de sucesso do diagrama de atividades (figura 4.3).

O cenário de sucesso deste diagrama contempla as seguintes atividades: "*Realizar download do arquivo de configuração (3.1)*", "*CFG: Interpretar comando do C&C (3.1.1)*", "*Enviar msg de sucesso ao C&C (3.1.2)*", "*Disparar temporizador para efetuar outro request (4)*", "*Enviar requisição DUPLA (5)*", "*Enviar requisição para download da atualização do malware (6)*", "*Realizar atualização (7)*", "*AT: Interpretar comando do C&C (8.1)*", "*Enviar mensagem de sucesso ao C&C (9)*", "*Disparar temporizador para a próxima execução da Fase Atualização (10)*", "*Keep-alive: Testar conectividade em intervalo de tempo predefinido (11)*", "*KA: Interpretar comando do C&C (12.1)*", "*Disparar tempo-*

rizador para a próxima execução da fase Keep-Alive (13)" e "*Interromper Fase Keep-Alive (14)*". As siglas `CFG`, `AT` e `KA` das atividades de interpretação de comando do C&C significam, Configuração, Atualização e Keep-Alive, respectivamente. Além do caminho do fluxo, foram utilizados números para indicar a ordem da ocorrência das atividades para facilitar a leitura do diagrama.

Essas atividades abrangem métodos concretos e abstratos no código do simulador. Alguns métodos abstratos foram implementados e poderão ser estendidos de acordo com a configuração que o *bot* deverá possuir.

A primeira atividade "*Realizar conexões à lista de C&C (1)*" se trata da fase Recém-Infecção que pode ser estendida, mas não está implementada neste simulador. Os motivos estão citados na descrição minuciosa dessa fase na Subseção 4.1.1, "Recém-Infecção". As outras atividades estão presentes no código em métodos abstratos, sendo alguns implementados.

Por este simulador tratar de fases da área de identificação e prevenção de atividades maliciosas, todas as mensagens do tipo `POST` que adicionam algum tipo de *payload* foram implementadas com poucos dados, por se tratar, apenas, do envio dos dados básicos do *bot*, como identificadores do *bot* e da *botnet*, seu sistema operacional, dentre outros. Dados roubados mais detalhados do *bot* foram desconsiderados. Sua montagem é configurável e sua adição, tanto após a abertura de conexão (no caso do `POST`), quanto na `URL` (caso do `GET`), é parametrizável de acordo com o arquivo de configuração do simulador.

Cada fase está implementada por um método da classe `Bot` que controla o sequenciamento, a repetição, a temporização e a ocorrência de cada uma delas. Métodos abstratos estão declarados e alguns implementados para desempenhar papéis configuráveis ou, apenas, diferentes na maneira de executar a funcionalidade, mas mantendo sua consistência. A geração do *trace* é contínua, conforme um *bot* o é na realidade.

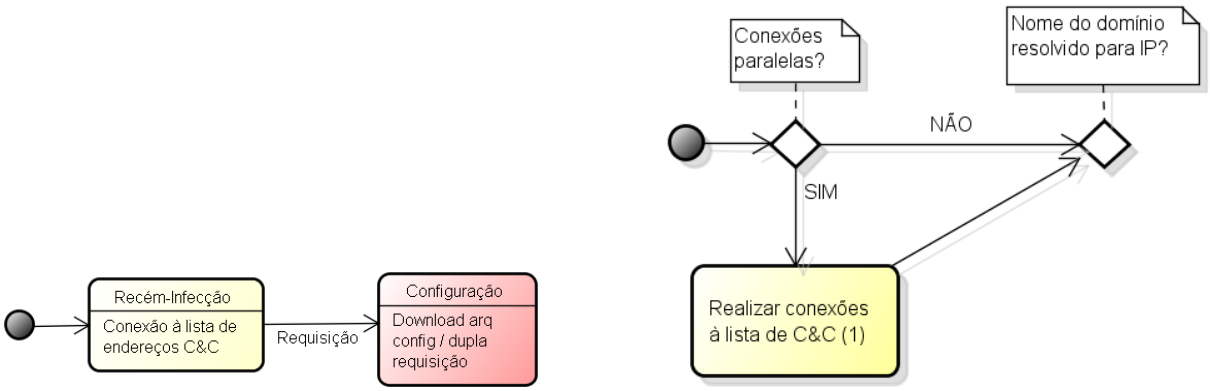
Adicionalmente, uma checagem de servidor pode ser feita antes de quaisquer requisições. Casos com comportamentos peculiares, como a escolha de outro domínio (PORRAS, 2009) ou o envio de *queries* `DNS` à procura do domínio não encontrado (SNORT, 2010), podem ser implementados. De acordo com o artigo (PORRAS, 2009), mensagens para obtenção de `IP externo` à rede do *bot* fazem parte do cenário da fase de propagação, em que ocorre o envio de uma requisição a um *site* ou a um C&C que retorna no corpo da resposta o endereço do `IP` do *bot* para que este seja acessado de fora de sua rede. Por não se tratar das fases modeladas neste trabalho, essas mensagens foram desconsideradas,

mas poderão ser extendidas em trabalhos futuros.

O tamanho e conteúdo das mensagens de resposta do C&C são configurados no servidor de comunicação do *bot* pelo usuário. Nesta modelagem, o servidor atua passivamente, provendo os arquivos requisitados ou respondendo com outros códigos de retorno, como os de redirecionamento ou acesso negado, por exemplo.

4.1.1 RECÉM-INFECÇÃO

É o ponto de partida da máquina de estados (figura 4.4(a)) e do diagrama de atividades (figura 4.4(b) - cor amarela).



(a) Estados (b) Atividades

FIG. 4.4: Diagramas da Transição Recém-infecção - Configuração

Nesta fase, a máquina se encontra recém-infectada. Pode ocorrer uma sequência de abertura de diversas conexões do *bot* aos canais de comunicação com os C&Cs da *botnet*, os quais o *bot* irá manter contato durante seu ciclo de vida. A lista de servidores C&C pode ser composta por `URL` ou `IP`. Esta lista pode ser atualizada com o tempo ou gerada dinamicamente através de um algoritmo próprio. Em ambos os casos, a lista e o algoritmo de geração de nomes de domínios são instalados junto com o *malware*.

O *bot* pode realizar tentativas de conexão a um ou mais servidores C&C de sua lista a fim de verificar se esta lista está atualizada. Estas tentativas de conexão primárias, sem a ocorrência de requisições para obtenção de arquivos ou envio de informações, também caracteriza a fase Recém-Infecção do *bot*.

Nesta e em qualquer outra fase, caso ocorra algum tipo de erro de nome de `DNS`, o *bot* poderá se comportar de várias formas dependendo de sua configuração. Um exemplo de comportamento é o de retirar o nome do servidor que retornou erro após um determinado

número de tentativas e/ou um determinado tempo. As tentativas de conexão podem ocorrer sequencialmente ou depois de um tempo configurado no *malware*.

Como esta fase trata de um cenário mais complexo com simultaneidade de conexões e, principalmente, por ser mais característico de *botnets* que utilizam o protocolo `IRC`, esta fase não foi implementada no simulador. A declaração abstrata do método que executa esta fase está presente no código, respeitando o sequenciamento de execução das fases.

4.1.2 CONFIGURAÇÃO

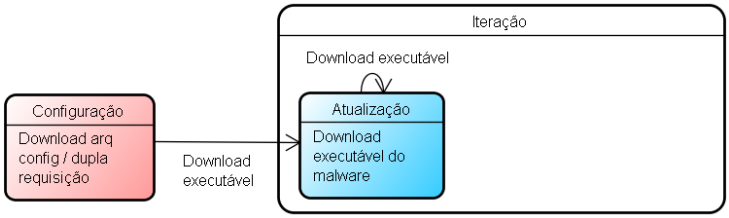
Fase opcional, pois o *bot* pode passar direto da Recém-Infecção para a Atualização. É definida por comportamentos que ocorrem uma única vez no ciclo de vida do *bot*, antes da primeira atualização do *malware*. Os fluxos de atividades desta fase estão ilustrados no diagrama de estados 4.5(a) e no diagrama de atividades 4.5(b) (cor rosa).

Foi identificado que o *bot* possui um arquivo que contém as configurações da *botnet* (SILVA, 2012) e (MATROSOV, 2011). Seu conteúdo pode incluir: lista de servidores (`IP` ou `URL`), algoritmo de formação de nomes de domínios (DGA), temporização das requisições do *bot*, entre outros comandos de configurações internas da máquina infectada e de roubo de dados. Casos de exceção de nome de domínio não resolvido para `IP` são tratados pela atividade "*Tratar nome do domínio não resolvido para IP (2)*" (figura 4.5(b)(2)) que possui método abstrato sem implementação.

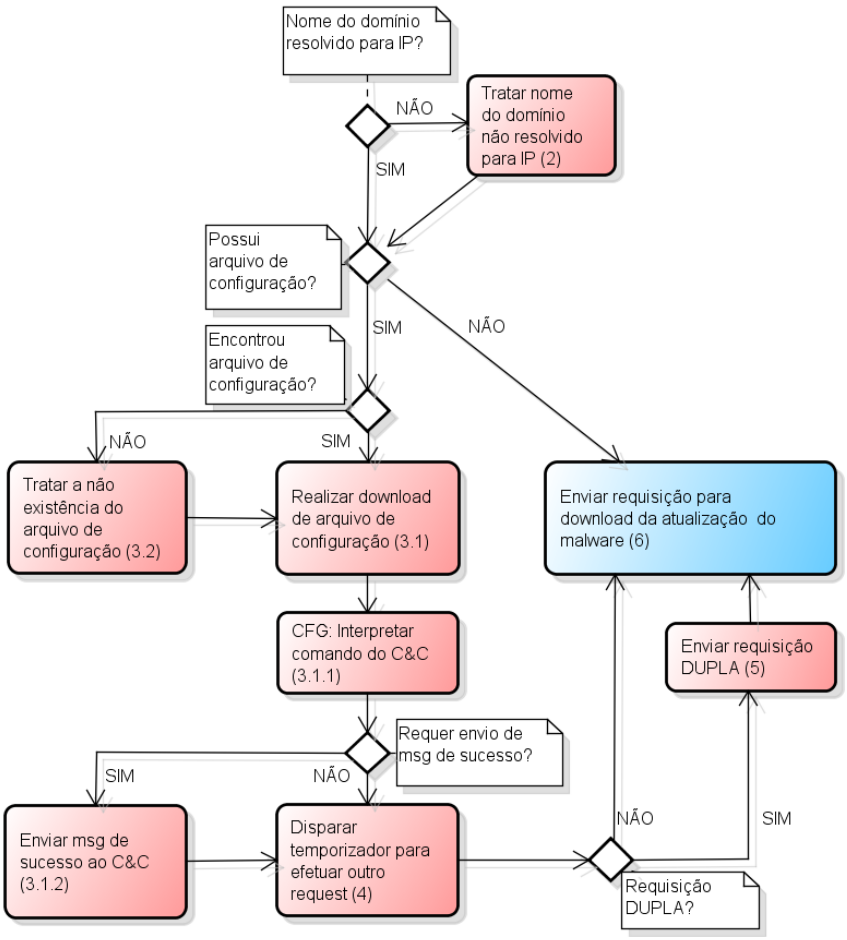
O arquivo de configuração pode ser obtido separadamente na fase Configuração do *bot* (BINSALLEEH, 2010) ou estar incluído no executável da infecção inicial da máquina (SILVA, 2012). Este arquivo de configuração pode ser atualizado. Isso varia a cada *botnet*. É nesta fase que ocorrem atividades para obtenção de arquivo de configuração no fluxo "*Realizar download de arquivo de configuração (3.1)*" (figura 4.5(b)(3.1)), e/ou dupla requisição (BINSALLEEH, 2010) (figura 4.5(b)(5)) antes da transferência do executável do *malware*. No diagrama de atividades, é mostrada a dupla requisição paralela que pode ser disparada imediatamente depois de um temporizador nos fluxos "*Enviar requisição DUPLA (5)*" e "*Disparar temporizador para efetuar outro request (4)*" (figura 4.5(b), itens (5) e (4), respectivamente). Optou-se por representar este fluxo opcional, pois uma família significativa de *bots*, a *Zbot*, possui essa característica de envio de dupla requisição que é caracterizada pelo envio de duas requisições, uma imediatamente após a outra, sem nenhuma resposta entre elas.

Já a requisição do arquivo ocorre com uma transação sequencial nos fluxos "*Realizar*

download de arquivo de configuração (3.1)", "*CFG: Interpretar comando do C&C (3.1.1)*" e "*Enviar msg de sucesso ao C&C (3.1.2)*" (figura 4.5(b), itens (3.1), (3.1.1) e (3.1.2), nesta ordem). Esses dois fluxos são alternativos, porque podem ocorrer independentemente um do outro, sendo que um deles deve obrigatoriamente existir nesta fase.



(a) Estados



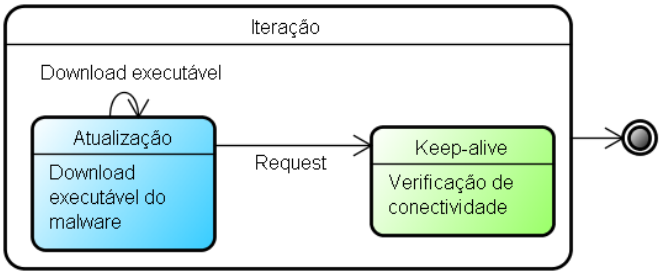
(b) Atividades

FIG. 4.5: Diagramas da Transição Configuração - Atualização

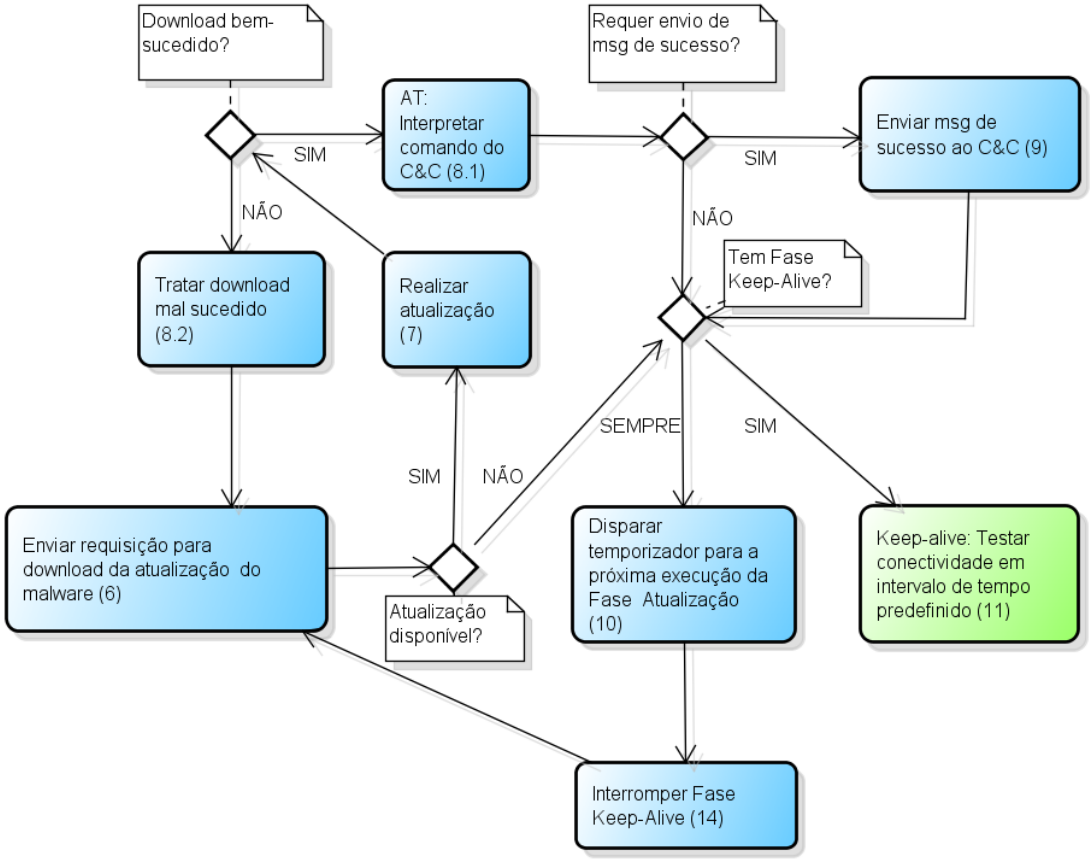
Caso o arquivo de configuração seja inexistente, o comportamento do *bot* pode variar de acordo com sua configuração. Ele pode tentar realizar requisições novamente, um determinado número de vezes, no mesmo servidor ou em outro. Esses comportamentos

de exceção estão abstratos no simulador e representados pela atividade "*Tratar a não existência do arquivo de configuração (3.2)*" (figura 4.3(3.2)) e podem ser implementados pelo usuário dessa ferramenta.

4.1.3 ATUALIZAÇÃO



(a) Estados



(b) Atividades

FIG. 4.6: Diagramas da Transição Atualização-Keep-Alive

Fase em que ocorre o *download* da versão mais atual do executável do *malware* (SILVA, 2012) (fluxo obrigatório), conforme o apresentado nas atividades "*Enviar requisição para*

download da atualização do malware (6)", "*Realizar atualização (7)*", "*AT: Interpretar comando do C&C (8.1)*", "*Enviar msg de sucesso ao C&C (9)*", "*Disparar temporizador para a próxima execução da Fase Atualização (10)*" e "*Interromper Fase Keep-Alive (14)*" na figura 4.6(a), itens(6), (7), (8.1), (9), (10) e (14), respectivamente, representados na cor azul, e no diagrama de estados (figura 4.6(b)).

O arquivo obtido na fase Configuração pode ser atualizado nesta fase. Já o arquivo que contém o *malware* é, geralmente, do tipo binário e diferente dos que são utilizados na fase Keep-Alive.

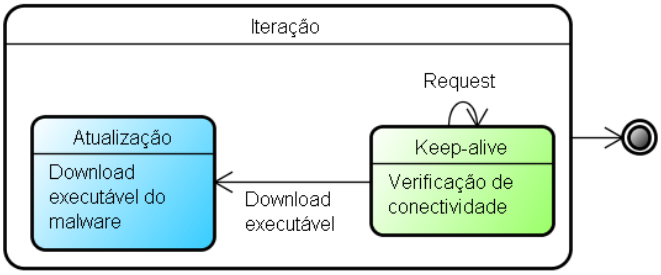
Caso o *bot* esteja programado para executar somente esta fase, como é o caso do Confiker (PORRAS, 2009), a periodicidade na comunicação do *bot* com o C&C caracteriza um comportamento de *keep-alive* do *bot*. Neste caso, apenas a busca por atualização é realizada. Nenhum tipo de envio de requisição com informação do *bot* é feita. Isso justifica a inexistência de fluxos separados de *keep-alive*. Na implementação, deve ser escolhido o número de requisições para atualização. No exemplo do Confiker, este número é 250. Após este número ter sido atingido, um temporizador que marca a próxima execução desta fase é disparado. Novamente, os tratamentos de exceção particulares de cada *bot* podem ser implementados na atividade "*Tratar download mal sucedido (8.2)*" (figura 4.3(8.2)).

Todas as transações ocorrem sequencialmente nesta fase. Quando o temporizador da fase Atualização termina seu ciclo, todas as atividades da fase Keep-Alive, caso exista, são suspensas, conforme atividade "*Interromper Fase Keep-Alive (14)*" (figura 4.3(10)), até que a fase Atualização termine todas as suas transações (requisição e realização de *download*, interpretação de comando e envio de mensagem de sucesso de *download*).

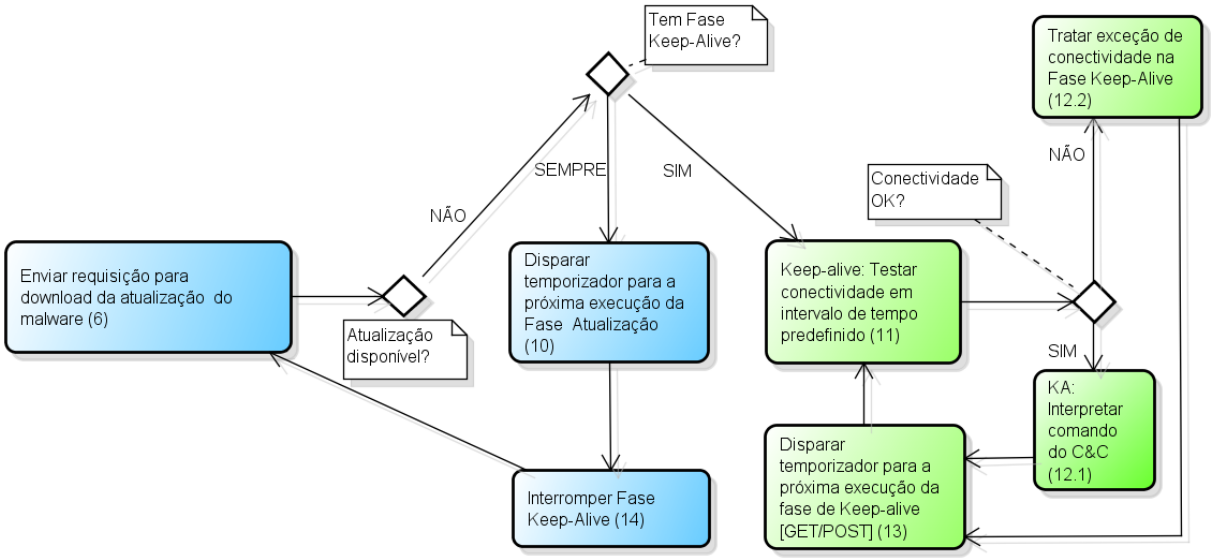
4.1.4 KEEP-ALIVE

O *bot* realiza a verificação de conectividade através de sua comunicação com os C&Cs em intervalos regulares configurados no *malware* (RODIONOV, 2011) e (MATROSOV, 2011), conforme as atividades "*Keep-alive: Testar conectividade em intervalo de tempo predefinido (11)*", "*KA: Interpretar comando do C&C (12.1)*" e "*Disparar temporizador para a próxima execução da fase de Keep-alive [GET/POST] (13)*" na figura 4.7(b), na cor verde, itens (11), (12.1) e (13). O *bot* informa que ele ainda pertence à *botnet*, ou seja, que ele ainda está ativo, através de requisições enviadas de tempos em tempos e quando a máquina infectada é reinicializada ou muda de rede.

No simulador, estão implementados os fluxos de *keep-alive* para os métodos `GET` e `POST`



(a) Estados



(b) Atividades

FIG. 4.7: Diagramas da Transição Keep-Alive - Atualização

separadamente na atividade "*Keep-alive: Testar conectividade em intervalo de tempo predefinido (11)*" (figura 4.7(b)(11)). O método `POST` é utilizado, geralmente, para o envio de dados roubados do *bot* ao C&C. Já o método `GET` pode ser utilizado tanto para o envio de informações do *bot* quanto para o teste de conectividade do *bot* ao C&C ou a outros domínios, neste caso, concomitantemente com o método `POST`. Os temporizadores dos métodos são diferentes e tratados pela atividade "*Disparar temporizador para a próxima execução da fase de Keep-alive [GET/POST] (13)*" (figura 4.7(b)(13)), caso ambos sejam utilizados. Então, optou-se por desmembrar o fluxo de *keep-alive* em dois, para o enriquecimento das funcionalidades.

Sempre que uma requisição é bem-sucedida, o método abstrato de tratamento de comando é acionado na atividade "*KA: Interpretar comando do C&C (12.1)*", conforme a figura 4.7(b)(12.1).

Casos de exceção na conexão do *bot* com o C&C e/ou a outros domínios (para checagem

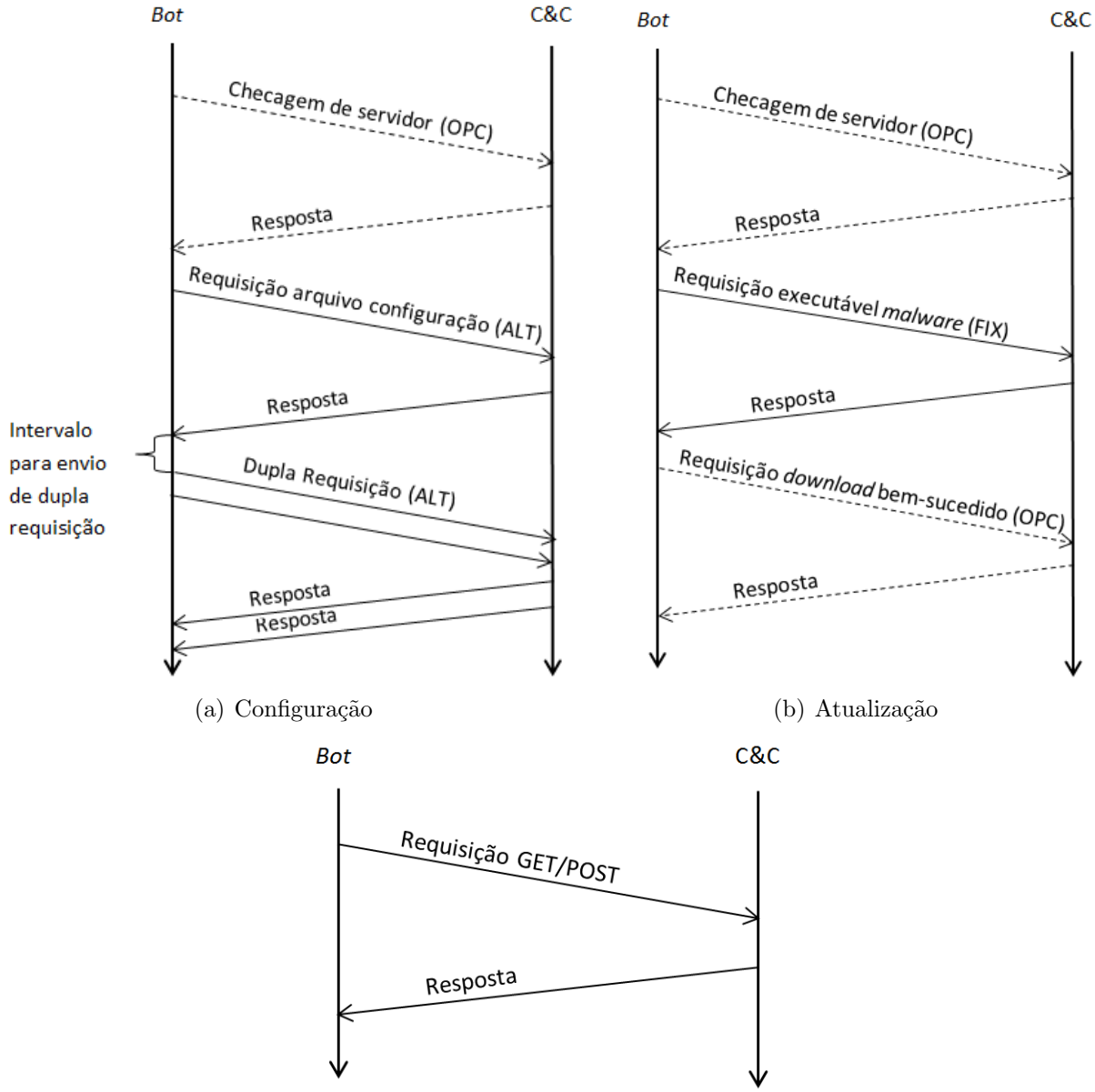
de conexão) são tratados pela atividade asbtrata e sem implementação "*Tratar exceção de conectividade na fase Keep-Alive (12.2)*", vide figura 4.7(b)(12.2).

Em (RICCARDI, 2013), as mensagens de *keep-alive* são classificadas como *log* e contém as principais informações de *status* do *bot*, como sua identificação, a de sua *botnet*, seu sistema operacional, dentre outras. A frequência da ocorrência dessas mensagens é maior que as do tipo *report* e o seu tamanho é menor. As mensagens do tipo *report* são maiores por conterem mais informações do *bot* e, por vezes, sofrem a fragmentação do pacote a ser enviado. Apenas as mensagens do tipo *log* estão representadas nesta fase, a fim de ser mantida a simplicidade do cenário que, posteriormente, pode ser enriquecido em trabalhos futuros.

4.1.5 RESUMO DAS ATIVIDADES DAS FASES DO BOT

A execução das fases pode ser representada através dos fluxos (ou transações) entre o *bot* e o C&C. A figura 4.8 ilustra os fluxos das atividades das fases Configuração (4.8(a)), Atualização (4.8(b)) e Keep-Alive (4.8(c)).

Os fluxos fixos são obrigatórios quando da existência da fase. Os opcionais podem ou não ocorrer. Já os alternativos são representados pela ocorrência obrigatória de, pelo menos, um fluxo. Todos os fluxos da fase Keep-Alive são obrigatórios.



(a) Configuração

(b) Atualização

(c) Keep-Alive

FIG. 4.8: Fluxo de execução das fases - OPC: Opcional / FIX: Fixo / ALT: Alternativo

5 SIMULAÇÃO

Um dos diferenciais deste trabalho é a maneira como as fases propostas foram implementadas no simulador. Isso, por si só, visa validar o modelo proposto. Os *traces* simulados são validados através de testes comparativos com *traces* disponibilizados, na Seção 6, "Experimentos e Resultados".

Métodos abstratos estão declarados e alguns implementados para desempenhar papéis configuráveis ou diferentes na maneira de executar as funcionalidades. Todas as fases estão implementadas. As características e a comunicação entre as fases, com as respectivas mensagem geradas, são descritas a seguir.

5.1 CONFIGURAÇÕES GERAIS DO SIMULADOR

Antes de cada requisição, pode ser configurado o envio de outra ao domínio escolhido, para fins de checagem de sua existência e de conectividade. Isso se justifica nos casos em que há algoritmos de geração de domínio (DGAs) que produzem numerosos nomes de domínios periodicamente. Esses algoritmos são amplamente utilizados na atualidade e estão sempre se modificando para evadir as técnicas de detecção. Um DGA forma o nome do domínio para onde a requisição será enviada. Essa geração dinâmica pode produzir centenas ou milhares de nomes por execução periódica antes do envio da requisição, como é o caso do Confiker (PORRAS, 2009). A implementação de um DGA pode ser feita nos pontos extensíveis do simulador nos métodos abstratos. Nenhum deles foi implementado porque trabalhou-se com apenas um servidor, visando manter o cenário simples. O foco é o *bot*, então, o funcionamento correto do *bot* com o cenário de um servidor, visa garantir a temporização, o sequenciamento e a ocorrência dos diálogos e das fases, que é o diferencial da modelagem deste trabalho.

O cenário com mais servidores implica, também, no estudo da distribuição geográfica desses servidores, tal como o é num cenário real. Como esses servidores podem estar em diversas localidades e, por entender que isso constitui um campo de estudo abrangente, optou-se em não tratar este tipo de tráfego com diversos servidores, o que tornaria o cenário de análise mais complexo para este trabalho. Tal assunto pode ser estudado em trabalhos futuros, a fim de enriquecer a qualidade e a abrangência dos *traces* de *bot*.

Técnicas de autenticação de conexão e criptografia da `URL` na requisição foram desconsideradas para a simplicidade do cenário ser mantida. Isso implicaria no estudo dessas técnicas para implementação e em utilização de conexão segura, o que nem sempre ocorre nas *botnets*. Essas implementações podem ser feitas em trabalhos futuros. Já a criptografia do *payload* pode ser efetuada a partir da concretização do método abstrato para esta finalidade.

Está declarada a classe "*Cipher*" que engloba os algoritmos de criptografia `AES` (Rijndael), `DES`, `RSA`, `RC4`, dentre outros. Tanto o nome do algoritmo quanto o tamanho da chave, em `bits`, são lidos do arquivo de configuração do simulador. Se mais de uma camada de criptografia quiser ser aplicada, é somente concretizar o método de criptografia de *payload*.

Somente os casos de sucesso de envio e recebimento de mensagens estão implementados para manter a simplicidade do cenário. Todas as requisições bem-sucedidas que possuam o código de retorno "`200 OK`", exceto as de envio de mensagem de sucesso e o envio de dupla requisição, disparam um tratamento de comando. Isso possibilita a interpretação e execução do comando que possa existir na resposta do C&C ao *bot*. Para este cenário configurável mais complexo, a implementação é requerida.

O simulador poderá ser configurado para tratar casos de exceção já previstos nas fases. Respostas com códigos "`3XX`" que indicam"*Not Modified*'', "*Moved Temporarily*", "*Moved Permanently*", dentre outros, estão configuradas para serem tratadas como requisições bem-sucedidas. Maiores desdobramentos deste comportamento, incluindo redirecionamentos, poderão ser implementados ou modificados.

O executável atualizado do *malware* pode redirecionar o *bot* para outro C&C. Em (FEILY, 2009), isso é chamado de migração de servidor e é útil para a estratégia do *botmaster* de evasão das conexões. Esse comportamento pode ser implementado no tratamento de comando.

Em (RICCARDI, 2013), os servidores são separados por objetivo de *download* (papéis): executável do *malware* ou arquivo de configuração. A escolha do servidor, por parte do *bot*, depende do tipo de requisição que o *bot* irá fazer. O papel dos servidores pode se sobrepor em apenas um. Apesar de não ser obrigatória esta separação de servidores, na inicialização do simulador é feita uma leitura de arquivos com o caminho absoluto dos arquivos nos servidores C&C de cada fase. A formação dos caminhos pode ser refinada e estendida com a utilização de algoritmos para este fim.

Durante a execução do *bot*, todas as mensagens indicativas de ações realizadas pelas fases são gravadas em um arquivo de *log*, com a extensão "`.txt`", que é atualizado com precisão na ordem de segundos. A checagem de servidor (configurável) utiliza o método `GET` e é feita com uma requisição somente para o nome do domínio, sem a complementação do caminho do arquivo no servidor.

Nas tabelas 5.1, 5.2 e 5.3 estão descritos os trinta e três campos do arquivo de configuração do simulador. Esse arquivo, no formato "`.txt`", é lido na inicialização do simulador e suas configurações são carregadas. Quaisquer erros de preenchimento dos campos deste arquivo abortam a execução do simulador. Esses campos representam os parâmetros de execução do *bot* em cada fase e alguns comportamentos gerais (comuns a todas as fases). Melhorias no carregamento e no formato destes arquivos para a configuração do simulador podem ser feitas em trabalhos futuros.

A temporização do *bot* ocorre nas fases Atualização e Keep-Alive, que são iterativas. Como a fase Atualização é obrigatória em qualquer *bot*, o controle do temporizador da fase Keep-Alive, através da classe `Timer`, é feito dentro do método que executa a fase Atualização. Isso é para garantir a correta temporização entre essas duas fases. Caso contrário, a temporização varia de acordo com o tamanho do arquivo transmitido e com a qualidade da conexão.

TAB. 5.1: Campos (parâmetros) do arquivo de configuração do simulador - parte 1

Campo	**Descrição**	**Tipo**
Identificador do *Bot*	Marca do *bot* que vai no campo "`Marca-do-Bot`" no cabeçalho do `HTTP`.	texto
Recém-Infecção	Existência da fase Recém-Infecção. No código há um método abstrato que executa esta fase. Ele é invocado antes da fase Configuração, respeitando o sequenciamento das fases. Este método está sem implementação.	boolean
Configuração	Existência da fase Configuração.	boolean
Atualização	Existência da fase Atualização.	boolean
Keep-Alive	Existência da fase Keep-Alive.	boolean
Atualização: mais de um arquivo para *download*	Indica se há mais de um arquivo para *download* na fase Atualização. Pode ser usado para controlar as datas das últimas atualizações desses arquivos.	boolean
Tempo de suspensão da Fase Atualização antes da Fase Keep-Alive	Tempo (em segundos) que a fase Keep-Alive executa antes de entrar a fase Atualização. Útil para os casos em que se deseja analisar o fluxo de Keep-Alive por um período de tempo antes de ocorrer a fase Atualização. O valor zero indica que não haverá suspensão da fase Atualização.	numérico
Configuração: arquivo de configuração	Existência de arquivo de configuração.	boolean
Configuração: método de requisição para arquivo de configuração	Caso haja requisição para o arquivo de configuração, este campo determina seu método de requisição.	`GET/POST`
Configuração-Atualização: arquivo configuração	Indica se o arquivo de configuração deve ser atualizado. Isso ocorrerá em toda execução da fase Atualização, exceto na primeira.	boolean
Configuração: mensagem de sucesso	Caso haja requisição para o arquivo de configuração, este campo determina a ocorrência do envio de mensagem de sucesso de seu *download*.	boolean
Configuração: dupla requisição	Indica a ocorrência de dupla requisição. Este fluxo é realizado depois da requisição para o arquivo de configuração. Ambos os fluxos podem coexistir na ordem descrita ou existir independentemente.	boolean
Configuração: método da dupla requisição	Caso haja dupla requisição, este campo determina o método utilizado.	`GET/POST`
Configuração: temporizador da dupla requisição	Caso haja dupla requisição, este campo determina o temporizador disparado antes de sua ocorrência.	numérico

TAB. 5.2: Campos (parâmetros) do arquivo de configuração do simulador - parte 2

Campo	**Descrição**	**Tipo**
Configuração: número de arquivos para *download*	Caso haja requisição para o arquivo de configuração, este campo determina a quantidade de requisições a este arquivo ou a mais arquivos, dependendo da implementação do método de recuperação de servidor da tabela de servidores da fase Configuração.	numérico
Atualização: temporizador	Indica o intervalo entre as execuções da fase Atualização.	numérico
Atualização: número de arquivos para *download*	Este campo determina a quantidade de requisições ao executável do *malware* (uma ou mais vezes) ou a mais de um arquivo, dependendo da implementação do método de recuperação de servidor da tabela de servidores da fase Atualização. `OBS`: No caso da implementação do *bot* Confiker, este campo indicará a quantidade de vezes que o mesmo arquivo deve ser requisitado.	numérico
Atualização: mensagem de sucesso	Este campo determina a ocorrência do envio de mensagem de sucesso de *download* do executável do *malware*.	boolean
Keep-Alive: existência do método de requisição `POST`	Indica a existência do fluxo desta fase com o método `POST`. Este fluxo pode coexistir com o do `GET`.	boolean
Keep-Alive: existência do método de requisição `GET`	Indica a existência do fluxo desta fase com o método `GET`. Este fluxo pode coexistir com o do `POST`.	boolean
Keep-Alive: temporizador mínimo do fluxo `GET`	Temporizador mínimo do fluxo `GET`. Um sorteio é feito entre os valores mínimo e máximo sempre antes da execução da fase Keep-Alive.	numérico
Keep-Alive: temporizador máximo do fluxo `GET`	Temporizador máximo do fluxo `GET`. Um sorteio é feito entre os valores mínimo e máximo sempre antes da execução da fase Keep-Alive.	numérico
Keep-Alive: temporizador mínimo do fluxo `POST`	Temporizador mínimo do fluxo `POST`. Um sorteio é feito entre os valores mínimo e máximo sempre antes da execução da fase Keep-Alive.	numérico
Keep-Alive: temporizador máximo do fluxo `POST`	Temporizador máximo do fluxo `POST`. Um sorteio é feito entre os valores mínimo e máximo sempre antes da execução da fase Keep-Alive.	numérico
Todas as fases: existência de *payload* na `URL` para o método `GET`.	Indica a adição de *payload* na `URL` da requisição com o método `GET`. É útil para os *bots* que não utilizam o método `POST` para requisição e necessitam enviar seus dados para o C&C.	boolean

TAB. 5.3: Campos (parâmetros) do arquivo de configuração do simulador - parte 3

Campo	**Descrição**	**Tipo**
Todas as fases: existência de *payload* na requisição	Indica a ocorrência de *payload* na requisição independente do método a ser utilizado. Para o método `POST`, após a abertura de conexão, os dados a serem transmitidos (*payload*) são escritos neste fluxo. Para o método `GET`, o *payload* vai na `URL` da requisição.	boolean
Todas as fases: criptografia de *payload*	Indica a existência de criptografia do *payload* na requisição. A classe `Cypher` foi declarada em método abstrato que pode ser estendido para diversos tipos de criptografia, inclusive com o uso de mais de um algoritmo e diversos valores de chave.	boolean
Todas as fases: algortimo de criptografia de *payload*	Indica o algoritmo de critptografia a ser utilizado.	texto
Todas as fases: chave do algortimo de criptografia de *payload*	Indica o número de `bits` da chave de critptografia a ser utilizada.	numérico
Todas as fases: dados do *payload*	Dados do *payload*. Esses dados podem ser utilizados em todos os *payloads* como uma padronização de seu tamanho ou podem ser reescritos nos métodos que montam o *payload* antes de transmití-lo.	texto
Todas as fases: checagem de servidor	Indica a existência de checagem do servidor escolhido antes da requisição para a `URL` da tabela de servidores de cada fase. Essa checagem está abstrata. Foi feita uma implementação para realizar uma requisição ao servidor (caminho da `URL` até a "/"), exceto para a fase Keep-Alive.	boolean
Todas as fases: persistência na conexão	Indica se mais tentativas de conexões devem ser feitas (persistentes) no caso de exceções na conexão. Isso é feito através de métodos abstratos de tratamento de erro/exceção de conexão.	boolean
Todas as fases: *timeout* da conexão	Indica o *timeout* da conexão. Este valor pode ser definido nos métodos de montagem de cabeçalho `HTTP`. O valor deste campo influencia diretamente na ocorrência de *reset* da conexão. O tamanho do *timeout* é inversamente proporcional à probabilidade de ocorrência de *reset* na conexão. O valor *false* neste campo, indica que o *timeout* padrão será utilizado.	boolean

5.2 IMPLEMENTAÇÃO DAS FASES DO *BOT*

A estrutura principal está na classe `Bot`. Nela, estão declarados os métodos que executam cada fase. A seguir, estão descritos alguns trechos da implementação das três fases, com alguns detalhes da implemetação no apêndice 9.1.

5.2.1 CONFIGURAÇÃO

A requisição para o arquivo de configuração é enviada e sua resposta é aguardada (fluxo sequencial). Já a dupla requisição é enviada com o uso de duas `threads`. O temporizador disparado antes dessa requisição invoca o método `sleep()` de ambas as `threads`.

A atualização do arqruivo de configuração do *bot* pode ser feita juntamente com a fase Atualização. Um dado do tipo *List* (para o nome do arquivo e para a data de sua última atualização) está declarado para os casos da existência de mais de um arquivo para ser atualizado. A quantidade de arquivos é configurada nos parâmetros do arquivo de configuração do simulador.

5.2.2 ATUALIZAÇÃO

Somente é inicializada após a fase Configuração. Quando é executada depois da fase Keep-Alive, todas as atividades e temporizadores de Keep-Alive (para ambos os métodos) são cancelados. Ao final da execução da fase Atualização, seu temporizador é disparado. Caso exista a fase Keep-Alive (conforme o arquivo de configuração do simulador), os temporizadores de Keep-Alive também são disparados. Após a primeira execução da fase Atualização, o arquivo da fase Configuração também pode ser adicionado à lista de arquivos a serem atualizados nesta fase.

No caso do *bot* Confiker, para que apenas uma transferência do arquivo mais atual seja realizada nas 250 requisições, a partir da segunda requisição, a data da última atualização é enviada no cabeçalho do protocolo `HTTP` no campo `If-Modified-Since` (SYSFORENSICS, 2012). Ele é preenchido com o valor do campo `Last-Modified` da última resposta bem sucedida da transferência deste arquivo. Esse tratamento é realizado em métodos abstratos de montagem de cabeçalho da mensagem e processamento de sua resposta. Ele está implementado dessa forma para que não haja mais de um *download* do mesmo arquivo requisitado sem atualização. No caso da existência de mais de um arquivo para ser atualizado (um ou mais arquivos de atualização e/ou configuração do *malware* - parametrizado

de acordo com o arquivo de configuração do simulador), o vetor de datas dos arquivos de configuração e de atualização da classe `Bot` pode ser utilizado para esse controle.

5.2.3 KEEP-ALIVE

É disparada uma `thread` por método de requisição a cada temporização atingida. Em outras palavras, os fluxos de *keep-alive* dos métodos `GET` e `POST` ocorrem independentemente um do outro. O código escrito desta fase é o mesmo para ambos os métodos.

Quando a requisição utiliza o método `POST`, pode ser adicionado um *payload* no campo `DATA` do protocolo `HTTP`. Este campo é preenchido com informações do *bot* com ou sem criptografia. Já quando o método utilizado é o `GET`, os dados do *payload* podem ser enviados apensos à `URL` da requisição.

Em alguns *bots*, o tempo de ocorrência de *keep-alive* pode estar dentro de um intervalo pequeno, de 4 a 6 segundos, por exemplo, para evadir técnicas de detecção. Isso é contemplado pelos campos de temporização mínimo e máximo de ambos os métodos na fase Keep-Alive. No simulador, a cada inicialização do temporizador da fase Keep-Alive, é feito um sorteio dentro do intervalo configurado neste arquivo. Caso os valores mínimo e máximo sejam iguais, a temporização será sempre a mesma, ou seja, fixa, pois o mesmo valor será retornado todas as vezes.

5.2.4 CONCLUSÃO DA IMPLEMENTAÇÃO

O simulador está implementado na linguagem `JAVA`, por ser bastante conhecida, possuir portabilidade entre diversos sistemas operacionais e por possuir um elevado número de bibliotecas que implementam funcionalidades das mais diversas. Está sendo utilizada a biblioteca `HTTPRequest` na implementação das requisições. Não foi utilizada a `libpcap` na implementação.

6 EXPERIMENTOS E RESULTADOS

Os experimentos foram conduzidos num ambiente contendo uma máquina cliente com o *bot* e outra máquina com um servidor *web* em rede externa à do *bot*. Como o mecanismo *pull* foi adotado neste experimento, toda iniciativa de conexão partiu do *bot* (cliente) para o C&C (servidor) que atuou passivamente, provendo os arquivos requisitados ou respondendo com o devido código nos casos de inexistência de arquivo. Na figura 6.1, é apresentada a arquitetura do experimento em rede externa à do *bot*.

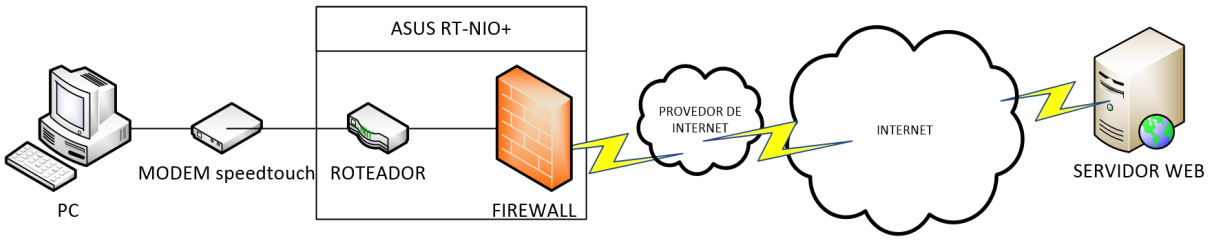


FIG. 6.1: Arquitetura dos experimentos em rede externa à do *bot*

Apenas um servidor foi utilizado para manter o cenário simples. Outros servidores podem ser adicionados com seus endereços devidamente configurados no simulador para o caso de aumento da complexidade no número de conexões e redirecionamentos.

De acordo com (RSA, 2012), o Zeus continua no topo das vendas de *malwares*. Vários artigos com diversas descrições desta *botnet* tratam de seu comportamento. Além disso, é uma representante da arquitetura centralizada, apesar de já ter variantes na versão `P2P` (STONE-GROSS, 2012). Portanto, seu *bot* será abordado neste trabalho nos experimentos, como um estudo de caso, devido a sua importância e à existência de seus *traces* disponibilizados em (SNORT, 2010).

Na Seção 6.1, "Experimentos com *Traces*", estão descritos os conteúdos de cada um desses *traces*. Eles são referenciados como *traces originais*. Os que são gerados pelo simulador são referenciados como *traces gerados*. Os *traces* originais disponibilizados pela Snort foram gerados por máquinas intencionalmente infectadas do `Sourcefire VRT` (*Vulnerability Research Team*), que é o grupo de pesquisa e resposta a incidentes de Segurança da Informação da `Snort`. A infecção utilizou vertentes conhecidas do *malware* Zeus, para fins de estudo do comportamento pós-infecção. As máquinas foram configuradas para

serem vulneráveis e terem instalação básica com o sistema operacional `Windows XP Service Pack 2`, sem *patches*.

A validação dos *traces* foi feita qualitativamente através da verificação da existência, do sequenciamento, da repetição e temporização dos diálogos entre o *bot* e o C&C. Também foi realizada a validação quantitativa com o uso de teste de aderência (DAGOSTINO, 1986) a fim de verificar a semelhança do comportamento do conjunto de dados do *trace* original e do gerado. O teste quantitativo foi utilizado para verificar estatisticamente a equivalência entre os *traces*.

Para isso, foram pesquisados trabalhos que utilizassem medidas de comparação entre tráfegos de rede `HTTP`. Comparação é um termo fundamental e, portanto, muito utilizado em trabalhos que tratam de detecção de tráfegos. Apesar do presente trabalho não se destinar à detecção de quaisquer tipos de tráfego, foram levadas em consideração as medidas mais comuns de comparação oriundas de trabalhos que tratam de detecção de tráfego de rede, para fins de validação dos experimentos.

Em (FREIRE, 2009), foram adotadas três medidas de comparação: intervalo entre requisições ao C&C (em segundos), tamanho da requisição e tamanho da resposta. Essas medidas foram utilizados para geração de distribuições empíricas para uma caracterização do tráfego `HTTP` considerado normal. Cada medida foi analisada separadamente e independentemente das outras. A utilização dessas três medidas combinadas para detecção no trabalho de (FREIRE, 2009) foi bem-sucedida.

Ainda em (FREIRE, 2009), são citados os trabalhos de Mah (MAH, 1997), Choi (CHOI, 1999) e Arlitt (ARLITT, 1995) para definição da escolha das medidas. Em (MAH, 1997), as três medidas escolhidas por Freire foram citadas e modeladas empiricamente para o tráfego `HTTP`. (CHOI, 1999) adotou a medida tamanho da requisição, dentre outras específicas para seu trabalho. (ARLITT, 1995) já utilizou o intervalo entre requisições. Com isso, Freire conclui que essas são as medidas mais comuns de características de tráfego de rede e também simples para utilização em seu trabalho. Diante do número de trabalhos que utilizaram essas medidas, da simplicidade de suas medições e da combinação dessas três medidas ter sido suficiente para comparação do tráfego `HTTP` em (FREIRE, 2009), no presente trabalho, essas três medidas são adotadas para fins de comparação dos *traces*.

Medidas definidas, então, foram pesquisadas as formas de comparação. Também em (FREIRE, 2009), é citado o trabalho de (ESTÉVEZ-TAPIADOR, 2004) que fez uso do teste de aderência `Kolmogorov-Smirnov` (`K-S`) (MASSEY, 1951) para comparação de

tráfegos. A justificativa da escolha deste teste se embasa em ser amplamente utilizado na área de detecção e em sua simplicidade, segundo (ESTÉVEZ-TAPIADOR, 2004). Este teste é não-paramétrico, ou seja, não assume previamente qual é a função de distribuição que os dados a serem analisados seguem.

Além do teste `K-S`, Freire também utiliza o teste `qui-quadrado` (COCHRAN, 1952). No presente trabalho, foi escolhido o teste `K-S` em sua forma completa devido ao seu bom comportamento com poucos dados em distribuições contínuas de uma única dimensão, diferentemente do teste `qui-quadrado`. Vale salientear que a modalidade de teste `K-S` utilizada pelo Freire e referenciada pelo Estévez-Tapiador et al. utilizou apenas uma amostra. Foram pesquisados desmembramentos deste teste e chegou-se ao encontro da modalidade com duas amostras, o que casa com o objetivo deste trabalho. Portanto, o teste `K-S` com duas amostras será referenciado, daqui por diante, como teste `K-S`.

O teste `K-S` (figura 6.2) compara dois conjuntos de dados independentes. São calculadas as funções de distribuição de probabilidade acumulada (`CDF`) para cada amostra em uma única dimensão, no caso, uma medida do *trace* original e a mesma medida do *trace* gerado. Após, é calculada a distância máxima (`D`) entre as duas funções. São comparados os pares de conjuntos de dados entre os *traces*. O objetivo deste teste é verificar a hipótese nula de igualdade entre as funções de distribuição acumulada de ambas as amostras de acordo com o grau de significância do teste. O grau de significância tem a ver com a probabilidade da hipótese nula ser rejeitada, quando verdadeira (erro do tipo I). Este grau é, geralmente, determinado em 5% (0,05) e assim foi adotado no presente trabalho.

Caso essa distância `D` ultrapasse um determinado valor, o teste é rejeitado. Do contrário, a hipótese nula de igualdade entre as funções de distribuição acumulada é aceita. Maiores detalhes de cálculos estão descritos em (YOUNG, 1977).

Para a realização do teste `K-S`, foi utilizada a ferramenta `SPSS Statistics`[9] da `IBM`, versão de teste. Foi escolhido o teste não-paramétrico `K-S` para amostras independentes. Os valores dos *traces* original e gerado para cada parâmetro em todos os experimentos estão no apêndice 9.2.

O histograma de cada um dos parâmetros dos *traces* original e gerado são mostrados no resultado de cada experimento. Adicionalmente, para o parâmetro intervalo entre requisições, é mostrado um gráfico, à parte, da medida de sequenciamento (ordem dos pacotes enviados) que é relevante para a modelagem e a simulação deste trabalho. Isso

[9] `IBM SPSS Statistics` http://www-01.ibm.com/software/analytics/spss/products/statistics/index.html

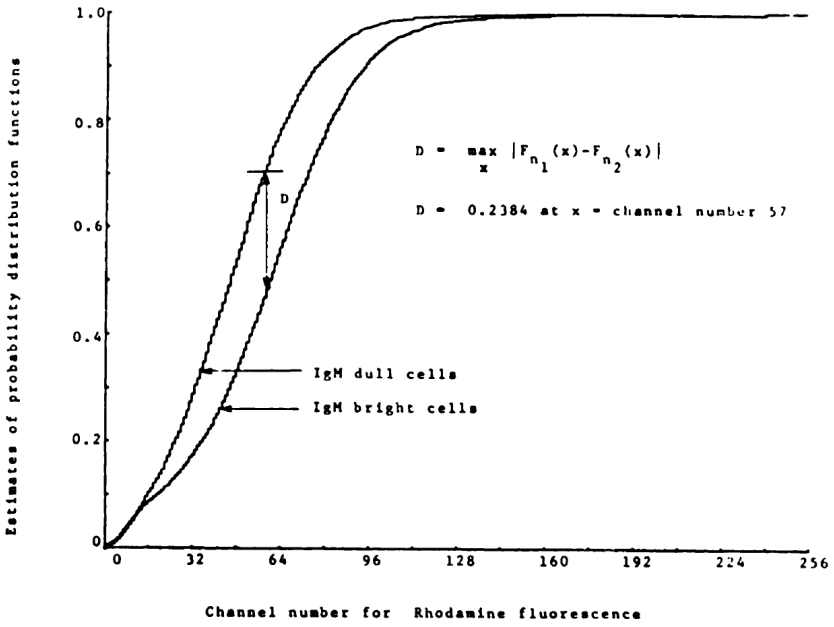


FIG. 6.2: Teste Kolmogorov-Smirnov para duas amostras independentes (YOUNG, 1977)

foi feito porque o teste K-S não leva em consideração o sequenciamento dos dados.

A medida *tamanho da resposta* pode ser configurada de acordo com o tamanho dos arquivos no servidor. Já o *tamanho da requisição* pode ser adequado, levando em consideração o tamanho do *payload* enviado e os campos do cabeçalho presentes no protocolo HTTP. Particularmente, a medida *intervalo entre requisições* fica à mercê da qualidade da conexão da rede onde o *bot* se encontra. Como esse simulador é executado em redes reais, o tráfego produzido por ele é sobreposto ao tráfego normal de rede. Com isso, a qualidade da conexão pode comprometer os temporizadores do *bot*.

Ao todo, foram realizados quatro experimentos individuais com apenas um *bot*. Três deles tiveram seus resultados comparados com *traces* disponibilizados e o outro experimento foi teórico, pois o *bot* foi configurado de acordo com descrições de artigos, sem *trace* a ser comparado. Os resultados obtidos mostram a corretude do funcionamento do *bot*.

Posteriormente, é descrito um cenário de um experimento da operação da *botnet*. Isso visa mostrar o funcionamento do conjunto de *bots* que formam a *botnet*. Apenas um servidor *web* foi adotado, pois o que está sendo experimentado é a corretude do funcionamento de vários *bots* por um período mais prolongado de tempo e a reação desses *bots* quando o servidor é desligado por um período e ligado novamente.

6.1 EXPERIMENTOS COM *TRACES*

Foram realizados três experimentos com o *bot* Zeus. A justificativa é devido à existência de *traces* do Zeus obtidos no site da Snort (*samples* 1, 2 e 3) (SNORT, 2010). Foram configuradas as funcionalidades deste *bot* no simulador, de acordo com cada *trace* disponibilizado. Os *traces* original e gerado apresentam o mesmo número de dados.

A seguir, o que está sendo levado em consideração em todos os experimentos para ser configurado no simulador:

- Nível de diálogo das transações, levando em consideração o método de requisição, a existência de *payload*, o sequenciamento e a temporização;
- Arquivos no servidor: pequenos arquivos `.txt` para método `POST`, executáveis para as fases de atualização de *malware* e de configuração com extensões `.exe`, com e sem compressão, e uso do método `GET`;
- Identificador do *bot* adicionado no cabeçalho `HTTP` no campo criado "Marca-do-Bot" com a sintaxe: "IME_N", em que "N" é um número natural que representa unicamente o *bot*. Isso facilita a filtragem do tráfego gerado.
- Apenas conexões dos protocolos `DNS` e `HTTP` foram consideradas, mas apenas as do `HTTP` foram analisadas;
- As requisições com o método `POST` foram configuradas com *payload* cujo tamanho foi baseado no do *payload* da requisição do *trace* original. Isso simula o envio de informações do *bot* para o C&C.

O que foi abstraído:

- O campo `HTTP referer` que pode constituir de atividades maliciosas que fogem do foco do trabalho. Um exemplo deste uso em ataque está descrito no relatório da Arbor (NETWORKS, 2012);
- Requisições com o método `GET` que fazem parte de outras fases (propagação, por exemplo);
- A criptografia do *payload* foi abstraída em todos os experimentos porque não influencia nos testes;

- Em todos os experimentos, a formação dos nomes de domínio, casos de exceção de conexão, de existência de servidor ou de arquivo não foram considerados, visando manter o cenário simples;

- Ocorrência de *reset* na conexão. De acordo com (STRAYER, 2008), a ocorrência de *reset* (`RST`) nas conexões indica que elas não foram completadas.

Em todos os experimentos, a formação dos nomes de domínio, incluindo as *queries* para obtenção da data e hora atuais (caso do Confiker), casos de exceção de conexão, de existência de servidor ou de arquivo não foram considerados, visando manter o cenário simples. Apenas conexões dos protocolos `DNS` e `HTTP` foram consideradas.

Foram analisadas a presença das fases Configuração, Atualização e Keep-Alive nos *traces* originais. As três fases foram encontradas e se apresentaram da seguinte forma:

- *Fase Configuração*: realiza o *download* de um arquivo. Após 30 segundos decorridos é enviada uma dupla requisição com o método `POST`, ambas para um segundo arquivo. Os três *traces* originais possuem essa fase com essas características;

- *Fase Atualização*: realização de *download* de apenas um executável com utilização do método `GET`. Logo após, é enviada uma mensagem com o método `POST`, indicando o sucesso deste *download*. Um ou mais *downloads* podem ser realizados nesta fase. Os *traces* 1 e 2 executam esta fase com o *download* de apenas 1 arquivo. Já o *trace* 3 realiza dois *downloads* de arquivos com nomes distintos e faz o envio de uma mensagem de sucesso com o método `POST` após o recebimento de cada arquivo;

- *Fase Keep-Alive*: realiza requisições ao mesmo arquivo em intervalos de tempo regulares de 5 segundos. Possui o mesmo tamanho de *payload*. Apenas o *trace* 1 possui essa fase. A última requisição com o método `POST` foi desconsiderada por possuir um tamanho diferente de *payload* das outras 9 requisições desta fase e, ainda, por somente ocorrer, aproximadamente, 60 segundos após a penúltima requisição com este mesmo método. Por existir essa divergência dessas características da última requisição com o método `POST` do restante do *trace*, isso foi considerado um indicativo de mensagem de envio de dados roubados, que não foi modelado neste trabalho e, portanto, desconsiderado.

Todas as respostas de todas as requisições dos três *traces* obtiveram o código "`200 OK`", indicativo de sucesso da requisição.

O tráfego foi capturado com a ferramenta `Wireshark`[10] e salvo em formato comum de *dump* de rede, o "`.pcap`"[11]. Após a captura ter sido feita, apenas o tráfego do *bot* foi filtrado e analisado. O campo "`Marca-do-bot`" foi importante na filtragem do tráfego `HTTP`. O tráfego filtrado foi analisado com o *trace* disponibilizado em cada experimento. O tamanho dos tráfegos disponibilizados é pequeno, por isso, o tamanho dos tráfegos gerados também o é. Dentre os três *traces* disponibilizados, o que tem o maior tamanho possui 14 requisições.

Essa filtragem se faz necessária, pois o simulador produz um tráfego de *bot* junto do tráfego de rede da máquina. Num cenário real, é exatamente isso que ocorre. O tráfego do *bot* fica misturado com o tráfego da máquina infectada. Nos experimentos, não foi levado em consideração o tráfego de fundo, pois os *traces* disponíveis para estudo continham apenas o tráfego malicioso. Por isso, para termos de comparação, apenas os tráfegos maliciosos foram levados em consideração.

Para cada experimento, são exibidos os histogramas, os valores da média, do desvio padrão e da mediana, e o resultado do teste `K-S` com duas amostras para os parâmetros: intervalo entre requisições, tamanho das requisições e tamanho das respostas. Os resultados apresentaram semelhança.

6.1.1 EXPERIMENTO 1 – ZEUS *TRACE* 1

O simulador foi configurado para executar as fases:

- *Configuração*: uma requisição para o arquivo de configuração com o método `GET`, espera de 30 segundos e envio de requisição dupla com o método `POST` para um segundo arquivo.
- *Atualização*: uma requisição para arquivo com o método `GET` e envio de mensagem de sucesso de *download*.
- *Keep-Alive*: 9 requisições para arquivos pequenos com uso do método `POST` com intervalo de 5 segundos entre as requisições. *Payload* de 118 `bytes`.

O cabeçalho das requisições `HTTP` foi configurado da seguinte forma:

[10] *Wireshark – What's on your network?* em http://www.wireshark.org/

[11] Formato de arquivo `libpcap` de captura de tráfego de rede em http://wiki.wireshark.org/Development/LibpcapFileFormat

- `Accept: */*`
- `Content-Type: t`
- `Pragma: no-cache`
- `Content-Length: <o tamanho do payload em bytes>`

Os campos "`User-Agent`", "`Host`" e "`Connection`" foram enviados automaticamente com seus valores padrão. O campo "`Marca-do-Bot`" foi suprimido para não aumentar o tamanho da requisição. A justificativa de se configurar o campo "`Content-Type`" com o valor "`t`" é, também, devido ao tamanho da requisição. Quando este campo não é configurado, o simulador o envia automaticamente com o valor padrão, o que aumenta o tamanho da requisição. Como o valor deste campo não influenciou a funcionalidade da requisição, que obteve o código de retorno "`200 OK`", optou-se por esta configuração com o valor "t" para minimizar o tamanho da requisição para ficar mais parecido com o valor do tamanho do *trace* original. A semelhança dos valores gerados com os originais pode ser percebida na coluna "Tamanho Requisição" da tabela 6.1 e na figura 6.3(b).

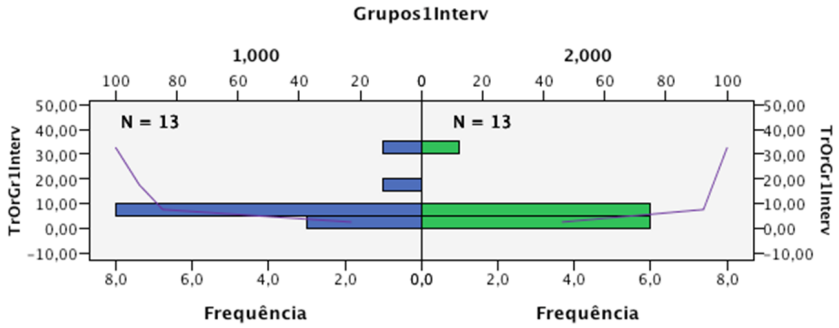
Este experimento contou com o maior número de dados a ser comparado, 13 para o parâmetro intervalo entre requisições e 14 para os outros dois. As três fases estiveram presentes neste experimento, o que possibilitou uma análise mais ampla na transição das mesmas.

Os tamanhos dos arquivos requisitados no *trace* gerado foram bem próximos daqueles do *trace* original. Isso foi decorrente da análise do tamanho do *payload* da resposta que representa o tamanho do arquivo transferido.

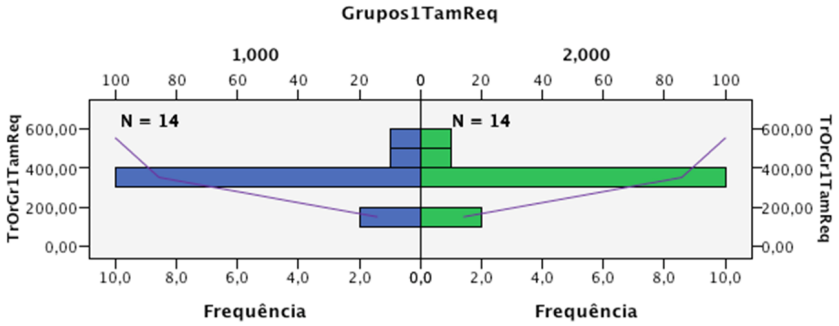
TAB. 6.1: Valores das medidas média, desvio padrão e mediana para os parâmetros dos *traces* original e gerado do experimento 1

Medida	**Trace**	**Intervalo Requisições**	**Tamanho Requisição**	**Tamanho Resposta**
Média	original	7,1056896	307,714285	145843,642857
Média	gerado	6,7177223	309,357143	83132,85714
Desvio padrão	original	7,992369	97,593685	543027,715994
Desvio padrão	gerado	7,845994	94,506497	308273,4958
Mediana	original	5,546322	307	469
Mediana	gerado	5,010306	307	469

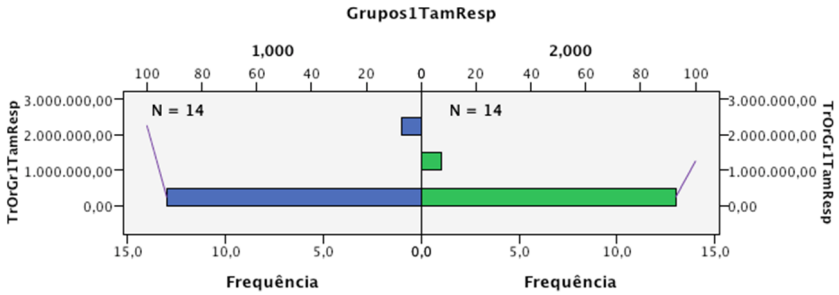
A qualidade da conexão interferiu no parâmetro intervalo entre requisições deste experimento. Apesar da temporização estar correta, o intervalo entre a requisição para atualização e a primeira requisição com o método `POST` (mesagem de sucesso) após a atualização levou cerca de 15 segundos no *trace* original. Para simular este comportamento, foram realizados *downloads* concorrentes com a execução do *bot* para que algum tipo de atraso ocorresse. E, assim, aconteceu. Houve atraso e as distribuições dos valores do *trace* gerado ficaram parecidas com as do *trace* original, como é mostrado nos resultados deste experimento.



(a) intervalo entre requisições



(b) tamanho das requisições



(c) tamanho das respostas

FIG. 6.3: Teste `K-S` com duas amostras: frequências e assintótica do experimento 1

Também ocorreu uma "quebra" dos intervalos regulares das requisições da fase Keep-Alive no *trace* original, que não se manteve fiel à temporização. No geral, o resultado final

do *trace* gerado foi significativo.

Todos os diálogos configurados ocorreram dentro de suas respectivas fases com o código de retorno "`200 OK`", conforme o esperado. Portanto, qualitativamente, as transações de ambos os *traces* original e gerado foram validadas.

Também foram calculados os valores de média, do desvio padrão e da mediana para os três parâmetros em todos os *traces* original e gerado. Os resultados estão na tabela 6.1.

Para todas as medidas, a hipótese nula de igualdade entre as funções de distribuição acumulada foi aceita. A frequência entre os grupos original (1,000, na cor azul), no lado esquerdo da figura, e gerado (2,000, na cor verde), no lado direito, são mostradas junto da assintótica na figura 6.3. O eixo horizontal significa a frequência dos valores de cada *trace* e o eixo vertical, os valores das amostras comparadas. Para cada medida, está ilustrado o histograma comparativo das duas amostras.

Os valores das medidas "tamanho das requisições" e "tamanho das respostas" são mais aproximados do *trace* disponibilizado, porque existe como "controlá-los". A medida "tamanho das requisições" está relacionada com o tamanho dos cabeçalhos das requisições e do *payload* transmitido. Tanto o cabeçalho quanto o *payload* são configuráveis no simulador. Isso faz com que esses tamanhos fiquem mais próximos daqueles do *trace* disponibilizado. Já a medida "tamanho das respostas" tem a ver com o tamanho dos arquivos no servidor e com o cabeçalho de suas respostas. O tamanho dos arquivos no servidor pode ser configurado independentemente de quaisquer configurações no simulador. Isso também faz com que os valores do *trace* gerado sejam próximos dos valores do *trace* obtido. Essa proximidade é refletida na semelhança entre as funções de probabilidade acumulada dos *traces* original e gerado comprovada pelo teste `K-S`.

6.1.2 EXPERIMENTO 2 – ZEUS *TRACE* 2

O simulador foi configurado para executar as fases:

- *Configuração*: uma requisição para o arquivo de configuração com o método `GET`, espera de 30 segundos e envio de requisição dupla com o método `POST` para um segundo arquivo.
- *Atualização*: uma requisição para um arquivo com o método `GET` e envio de mensagem de sucesso de *download*.

O cabeçalho das requisições `HTTP` foi configurado da seguinte forma:

- `Pragma: no-cache`
- `Content-Length: <o tamanho do payload em bytes>`
- `Marca-do-Bot: IME_1`
- `User-Agent: <Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1; SV1)>` ou `<valor padrão>`
- `Connection: Close ou keep-alive`

Os campos "`Host`" e "`Connection`" foram enviados automaticamente com seus valores padrão. Todas as requisições receberam o código de retorno "`200 OK`". Os tamanhos dos arquivos requisitados no *trace* gerado foram bem próximos daqueles do *trace* original.

A qualidade da conexão não interferiu na medida intervalo entre requisições deste experimento 2. A temporização se apresentou correta.

TAB. 6.2: Valores de média, desvio padrão e mediana para as medida dos *traces* original e gerado do experimento 2

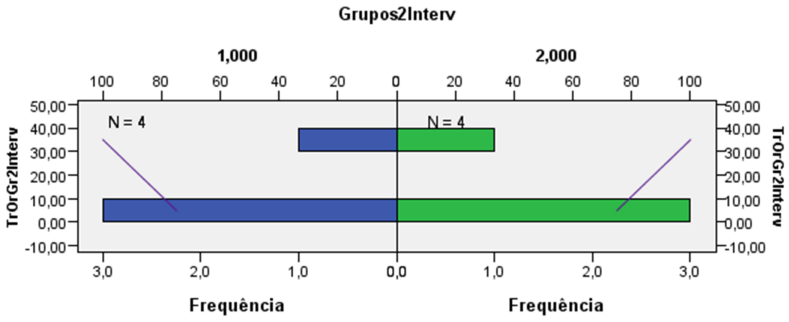
Medida	**Trace**	**Intervalo Requisições**	**Tamanho Requisição**	**Tamanho Resposta**
Média	original	8,007964	339,8	11316,6
Média	gerado	7,662644	362	12082,2
Desvio padrão	original	14,780900	171,193457	16074,792263
Desvio padrão	gerado	15,091247	119,816526	17005,389945
Mediana	original	0,932449	370	285
Mediana	gerado	0,15445	355	321

Os valores de média e desvio padrão e mediana para as três medidas estão na tabela 6.2.

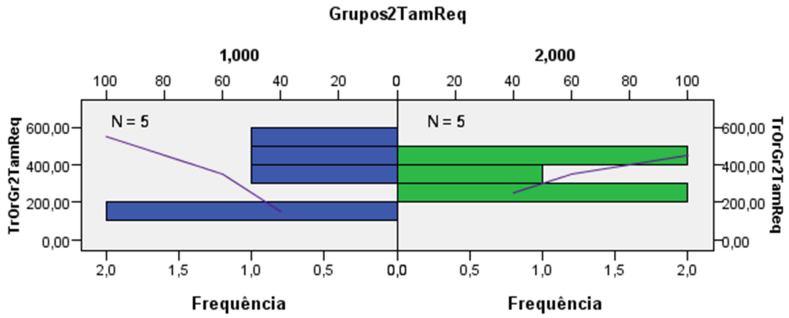
Este foi o experimento com o menor número de dados a ser comparado. Ao todo 4 para a medida intervalo entre requisições e 5 para as outras duas. O período de execução deste experimento para obtenção desses valores foi curto, o que permitiu com que a amostra tivesse os intervalos entre requisições parecidos.

Todos os diálogos configurados ocorreram dentro de suas respectivas fases, conforme o esperado. Portanto, qualitativamente, as transações de ambos os *traces* original e gerado foram validadas.

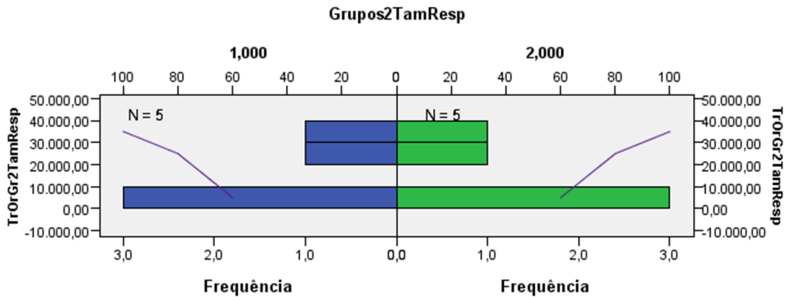
Em relação à analise quantitativa, primeiramente, os histogramas das medidas são mostrados na figura 6.4. O experimento 2 apresentou bastante semelhança nos percentu-



(a) intervalo entre requisições



(b) tamanho das requisições



(c) tamanho das respostas

FIG. 6.4: Teste `K-S` com duas amostras: frequências e assintótica do experimento 2

ais de comparação dos histogramas em duas das três medidas. Isso está mostrado nas figuras 6.4(a) intervalo entre requisições e 6.4(c) tamanho das respostas. A medida tamanho da requisição (figura 6.4(b)), apesar de discrepante, não interferiu no resultado final de aceitação da hipótese nula de igualdade entre as funções de distribuição acumulada. Para todas as medidas, a hipótese nula foi aceita.

6.1.3 EXPERIMENTO 3 – ZEUS *TRACE* 3

O simulador foi configurado para executar as fases:

- *Configuração*: uma requisição para o arquivo de configuração com o método `GET`,

espera de 30 segundos e envio de requisição dupla com o método `POST` para um segundo arquivo.

- *Atualização*: duas requisições para arquivos distintos com o método `GET` e envio de mensagem de sucesso de *download*.

O cabeçalho das requisições `HTTP` foi configurado da seguinte forma:

- `Accept: */*`
- `Pragma: no-cache`
- `Content-Length: <o tamanho do payload em bytes>`
- `Content-Type: text/plain [tipo do payload]`
- `Marca-do-Bot: IME_1`
- `User-Agent: Mozilla/4.0 (compatible; MSIE 6.0; Windows NT 5.1; SV1)`
- `Connection: Close ou keep-alive`

TAB. 6.3: Valores de média, desvio padrão e mediana para as medidas dos *traces* original e gerado do experimento 3

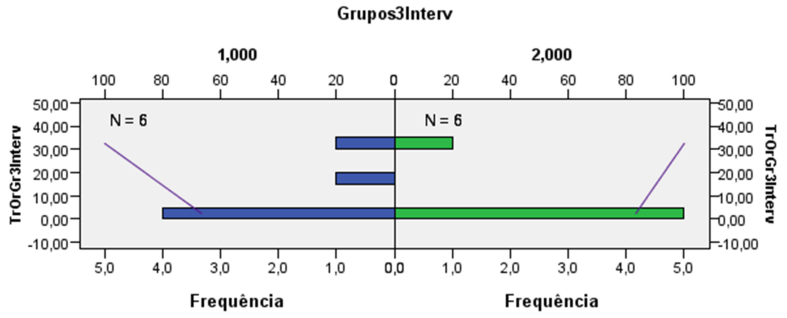
Medida	**Trace**	**Intervalo Requisições**	**Tamanho Requisição**	**Tamanho Resposta**
Média	original	8,517711	326,571428	128671,857142
Média	gerado	5,753812	370,285714	125669,714285
Desvio padrão	original	12,208303	158,543219	218152,444377
Desvio padrão	gerado	12,377676	142,120505	215254,453825
Mediana	original	2,1645565	371	488
Mediana	gerado	0,615577	419	321

Os campos "`Host`" foram enviados automaticamente com seus valores padrão. Todas as requisições receberam o código de retorno "`200 OK`". Os tamanhos dos arquivos requisitados no *trace* gerado foram bem próximos daqueles do *trace* original. A qualidade da conexão não interferiu significativamente na medida intervalo entre requisições deste experimento 3. A temporização funcionou corretamente.

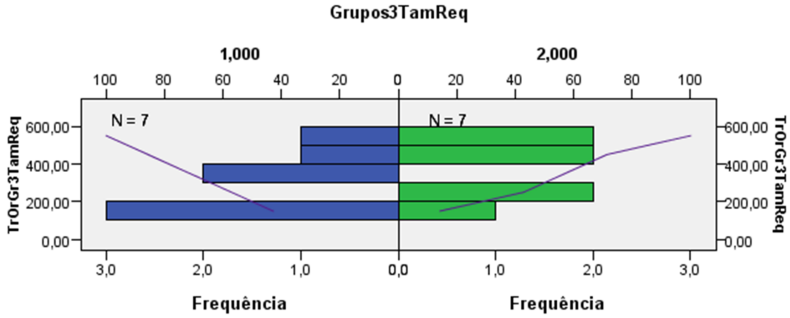
O experimento 3 apresentou bastante semelhança nos percentuais de comparação no histograma da medida tamanho das respostas (figura 6.5(c)).

Os valores de média e desvio padrão e mediana para as três medidas do experimento 3 estão na tabela 6.3.

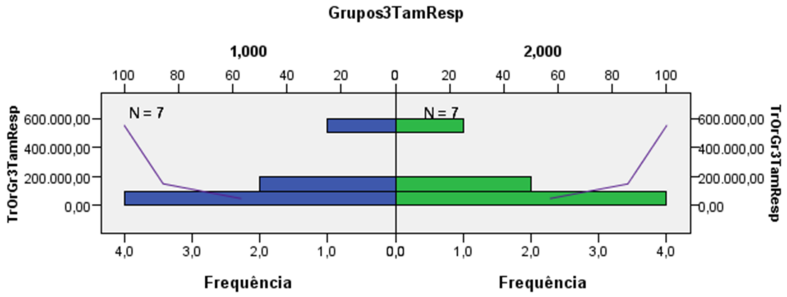
Todos os diálogos configurados ocorreram dentro de suas respectivas fases, conforme o esperado. Portanto, qualitativamente, as transações de ambos os *traces* original e gerado foram validadas.



(a) intervalo entre requisições



(b) tamanho das requisições



(c) tamanho das respostas

FIG. 6.5: Teste `K-S` com duas amostras: frequências e assintótica do experimento 3

Em relação à análise quantitativa, primeiramente, os histogramas das medidas são mostrados nas figuras: 6.5(a) intervalo entre requisições, 6.5(b) tamanho das requisições e 6.5(c) tamanho das respostas.

Novamente, a medida "intervalo entre requisições" teve um valor, um tanto quanto, discrepante. Isso pode ser percebido no histograma dessa medida na figura 6.5(a), em que

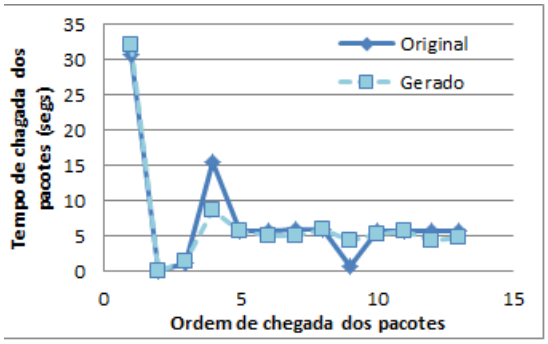
há uma barra do *trace* original sozinha no meio do gráfico. Provavelmente, é devido ao maior tempo na transmissão do arquivo na segunda requisição da fase Atualização, pois o segundo arquivo (cerca de 500B) trasmitido é maior que o primeiro (cerca de 100B). O *trace* gerado relativo a esse tráfego está representado na primeira barra inferior, que é maior que a do *trace* original.

A medida "tamanho das respostas" apresentou valores quase idênticos em ambos os *traces*, conforme o mostrado na figura 6.5(c). Já o "tamanho da requisição" foi a mais discrepante, porém, não levou à rejeição da hipótese nula do resultado do teste K-S.

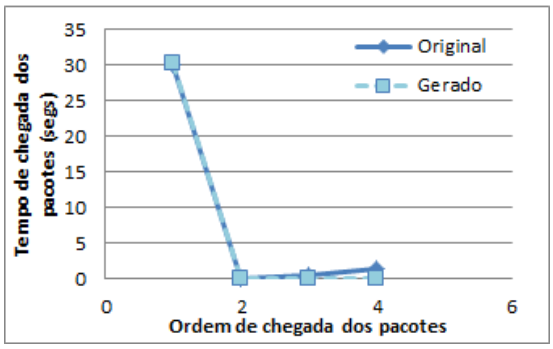
A aplicação do teste K-S com duas amostras comprova a semelhança dos valores apresentados. Em todos os resultados, para todas as medidas, a hipótese nula foi aceita.

6.1.4 RESULTADOS COMUNS A TODOS OS EXPERIMENTOS

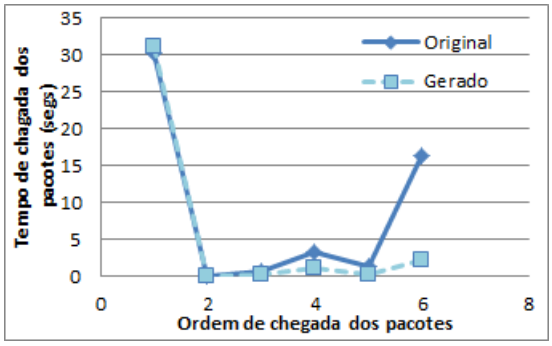
Em todas as requisições foram utilizadas portas randômicas altas acima de 60 mil. Todas as portas destino foram iguais a 80. Ao todo, apenas uma resolução de nome foi feita, pois todas as requisições foram feitas para o mesmo servidor, o que aponta a existência de tráfego DNS antes do início das requisições, de acordo com o esperado.



(a) Experimento 1



(b) Experimento 2



(c) Experimento 3

FIG. 6.6: Sequenciamento do intervalo entre requisições

A ordem é desprezada no teste K-S. Então, para comprovar o sequenciamento, foi construído um gráfico para cada experimento de acordo com o intervalo entre as requisições na ordem em que elas ocorreram, como é mostrado na figura 6.6. Os resultados estão em conformidade com esperado e todos os requisitos foram devidamente validados.

Todas as fases previstas nos cenários foram geradas. Os resultados foram significativos e refletidos na aceitação da hipótese nula de igualdade entre as funções de distribuição acumulada dos *traces* original e gerado para as três medidas analisadas, verificado pelo teste K-S para duas amostras independentes. Isso significa que ambas as funções apresentam o mesmo comportamento, cumprindo com o objetivo do teste na etapa de validação.

6.2 EXPERIMENTO TEÓRICO

Este cenário foi minimizado por se tratar de apenas 1 servidor e 1 *bot*. Foram escolhidas as seguintes configurações de acordo com o artigo "*A Foray into Confiker´s Logic and Rendezvous Points*" (PORRAS, 2009):

- Fase de Atualização
- 250 rodadas de atualização (no artigo) representadas (minimizadas) por 25 rodadas (no experimento)
- Checagem do servidor feita antes do envio da busca pelo arquivo executável do *malware* porque primeiro é tentado resolver o domínio para um IP.
- Temporização da rodada de 2 horas (no artigo) minimizada para 2 minutos (no experimento)
- Apenas requisições com o método GET utilizadas
- Informações sobre o *bot* enviadas na URL das requisições como apensos à URL. Neste cenário, será usado "q=0" porque nenhuma máquina foi infectada por este *bot*. Então, o *payload* na URL é: **?search?q=0**
- Arquivos no servidor com até 512 *bytes*.
- Servidor com domínio ".com".

Após o primeiro *download*, as requisições posteriores para o mesmo executável foram feitas com a adição da data da última atualização do arquivo no campo "`If-Modified-Since`" do cabeçalho `HTTP`. Isso faz com que o *download* ocorra somente em casos de novas atualizações do *malware* (arquivo) no servidor.

O cenário foi 100% validado. Apenas na primeira requisição ocorreu a transferência do arquivo executável, representando o *malware*. As demais obtiveram o código "`304 - Not Modified`", indicando que não houve atualização desde o último *download*, conforme o previsto.

Ao todo, houve 25 requisições com o método `GET` para obtenção do arquivo executável do `malware`, com o intervalo de 2 minutos para a próxima sequência de 25 requisições. Essa sequência de requisições ocorreu 3 vezes, ou seja, 3 ciclos da fase Atualização foram executados. Em todas as requisições, o *payload* "`?search?q=0`" esteve presente na `URL`.

Antes de cada requisição, foi feito um teste de conexão ao nome do servidor, também com o método `GET`. Apenas os fluxos da fase Atualização ocorreram. Portanto, todos os requisitos deste cenário foram devidamente validados.

6.3 EXPERIMENTO DA OPERAÇÃO DO CONJUNTO DE *BOTS*

Este experimento focou no funcionamento do conjunto de *bots*, ou seja, a *botnet*. Para isso, instanciou-se um simulador em cada máquina. Ao final, foi verificado o comportamento coletivo e se atividades foram executadas corretamente. Com isso, foi testada a viabilidade da solução e a corretude na implementação do simulador.

Ao todo, foram utilizados 10 *bots*, cada um com seu identificador único no campo "Marca-do-bot" no cabeçalho `HTTP` das requisições, numerados de 1 a 10. Apenas um servidor foi utilizado para hospedar os arquivos requisitados pelos *bots*.

A arquitetura do laboratório desse experimento conta com os seguintes elementos:

- 3 máquinas fisicamente idênticas que hospedam máquinas virtuais (`MVs`) que rodam individualmente os *bots* (em duas máquinas) e o servidor *web* dedicado a esse experimento (na terceira máquina separada das que estão os *bots*)
- As duas máquinas reais que hospedam os *bots* possuem 5 `MVs` cada, totalizando 10 `MVs`. Cada `MV` roda 1 *bot* com um endereço `IP` único.
- O tráfego de cada *bot* e do servidor foram capturados com a ferramenta `Wireshark` e salvo em formato comum de *dump* de rede com a extensão "`.pcap`".

- Os endereços IP dos *bots* são: 192.168.7.21 a 25 para a máquina real 2 e 192.168.7.31 a 35 para a máquina real 3.
- O servidor *web* roda na máquina real 1 com IP 192.168.7.1.
- Os *bots* e o servidor estão na mesma rede (192.168.7.0 com máscara 255.255.255.0).

Maiores detalhes sobre as configurações de máquinas físicas e virtuais estão no apêndice 9.3.

Após a montagem deste laboratório, um roteiro foi seguido para a execução do experimento:

1 Configurar todos os *bots* com os mesmos parâmetros do arquivo de configuração, exceto o campo "Marca-do-bot", conforme o descrito no apêndice 9.4. As identificações foram de IME_1 a IME_10.

2 Iniciar o servidor *web* com o comando "Put Online".

3 Iniciar a gravação dos *traces* nas 3 máquinas reais por 35 minutos.

4 Inicializar os *bots* em momentos distintos, simulando uma contaminação na rede com uma pequena diferença de tempo na ordem de segundos a poucos minutos, conforme o ilustrado na tabela 6.4.

5 Verificar o tráfego gerado por cada *bot* e pelo servidor.

6 Parar a gravação dos *traces* e salvá-los.

7 Inicializar a gravação dos *traces* nas 3 máquinas reais novamente.

8 Tirar o servidor do ar com o comando "Put Offline" por 5 minutos.

9 Verificar novamente o tráfego gerado por cada *bot* e pelo servidor. Dessa vez, são esperadas mensagem de erro com o código "403 - Forbidden". Isso é porque os *bots* não irão mais conseguir contactar o servidor.

10 Reiniciar o servidor com o comando "Put Online".

11 Verificar novamente o tráfego gerado por cada *bot* e pelo servidor. O tráfego deve estar normalizado e o código "403 - Forbidden" não deve mais existir.

12 Gravar os *traces* por mais 5 minutos, totalizando 10 minutos de gravação sendo 5 minutos com o servidor fora do ar e mais 5 após o retorno do servidor.

13 Aguardar a execução do experimento por aproximadamente 44h.

14 Verificar novamente o tráfego gerado por cada *bot* e pelo servidor (se todos continuaram operacionais).

15 Gravar os *traces* nas 3 máquinas reais por 30 minutos.

Apenas os tráfegos `HTTP` do *bot* e do servidor foram filtrados e analisados. Quaisquer tráfegos de rede diferentes disso foram desprezados. O campo "`Marca-do-bot`" foi importante na filtragem do tráfego `HTTP`.

Na tabela 6.4, estão destacados os tempos de inicialização dos *bots*, na coluna "`Inicialização`" e o intervalo entre essas inicializações, na coluna "`Intervalo`", como se fosse a diferença de tempo da contaminação nos *bots*. Os tempos das inicializações foram aleatórios.

TAB. 6.4: Tempos de início e de intervalo aproximados entre as inicializações dos *bots*

ID e IP do *Bot*	**Inicialização**	**Intervalo**
ID 1 IP .21	0"	0"
ID 2 IP .22	1'	1'
ID 3 IP .23	1'30"	30"
ID 4 IP .24	2'	30"
ID 5 IP .25	3'	1'
ID 6 IP .31	4'	1'
ID 7 IP .32	4'15"	15"
ID 8 IP .33	5'30"	1'15"
ID 9 IP .34	7'	1'30"
ID 10 IP .35	10'	3'

Os critérios analisados e verificados nesses tráfegos englobam o sequenciamento e a temporização dos fluxos de comunicação entre os *bots* e o servidor; a semântica de cada fluxo, incluindo a existência de *payload*, funcionamento pleno do *bot* por um período maior de execução e após a queda e reinicialização do servidor.

6.3.1 RESULTADOS DO EXPERIMENTO DA OPERAÇÃO DA *BOTNET*

Ao todo, 3 arquivos de *traces* foram gravados em cada máquina real. Estes arquivos representaram os 3 momentos distintos da execução dos *bots*:

- Inicialização dos bots. Duração da captura: 35 minutos.
- Interrupção do servidor por 5 minutos e sua reinicialização. Duração da captura: 10 minutos.
- Execução depois de, aproximadamente, 44h. Duração da captura: 30 minutos.

Os *traces* capturados englobaram o tráfego normal, comum a qualquer rede. As análises foram feitas após uma filtragem por protocolo `HTTP` e por `IP`.

Todos os *bots* continuaram sua execução normal após a queda e o retorno do servidor. Os códigos de sucesso esperados na fase Configuração, na primeira execução da fase Atualização e nas requisições da fase Keep-Alive ocorreram, indicando que os bots operaram corretamente. Nas demais execucões da fase Atualização, o código "`304 - Not Modified`" esteve presente nas respostas, conforme o esperado, pois o campo "`If-Modified-Since`" foi enviado a partir da segunda requisição desta fase.

O código "`403 - Forbidden`" ocorreu como resposta a todas as requisições feitas por todos os *bots* durante o período que o servidor esteve fora do ar (5 minutos). Após o retorno do servidor, este código não mais ocorreu.

Todos os diálogos esperados estiveram presentes:

- **Fase Configuração:** houve a requisição de arquivo de configuração com o método `GET` e seu *download*. Envio de dupla requisição com o método `POST` após 30 segundos do *download* do arquivo de configuração. Recebimento do código "`200 - OK`" em todas as respostas. O *payload* foi transmitido.
- **FaseAtualização - primeira execução:** ocorreu a requisição de arquivo que representa o *malware* e seu *download*. Envio de mensagem de sucesso com o método `POST` com transmissão de *payload* após o *download* bem-sucedido. Recebimento do código "200 - OK" em todas as respostas.
- **FaseAtualização - demais execuções:** Após 120 segundos, foi enviada uma requisição de arquivo que representa o *malware* com adição do campo "`If-Modified-Since`" e recebimento do código "`304 - Not Modified`" em todas as respostas.
- **Fase Keep-Alive - método GET:** Após 30 segundos da finalização da fase Atualização e a cada 30 segundos até a próxima execução da fase Atualização, ocorreu

uma requisição de arquivo com o método `GET` e seu *download*. Recebimento do código "`200 - OK`" em todas as respostas.

- **Fase Keep-Alive - método POST:** Após 30 segundos da finalização da fase Atualização e a cada 30 segundos até a próxima execução da fase Atualização, ocorreu uma requisição de arquivo com o método `POST` e seu *download*. Recebimento do código "`200 - OK`" em todas as respostas. O *payload* foi transmitido.

O experimento foi executado por 44 horas. Todos os *bots* e o servidor funcionaram corretamente indicando que a operação da *botnet* foi bem-sucedida. A continuidade da execução dos *bots* foi verificada.

O que pode ser observado foi uma carga inicial maior da rede, devido a todos os *bots* estarem realizando seu primeiro *download* de um arquivo de, aproximadamente, `2MB`. Com isso, o temporizador da dupla requisição, configurado para 30 segundos, permaneceu assim, apenas, no primeiro *bot* inicializado. Os demais tiveram atrasos de 180s a 269s. Findados os *downloads*, as temporizações voltaram ao normal e assim permaneceram por até dois dias depois, no final do experimento.

Portanto, para se simular uma *botnet*, o cenário com um ou mais servidores *web* e o instanciamento deste simulador pode ser posto em prática em diversas máquinas.

7 CONSIDERAÇÕES FINAIS

Este trabalho contribui com a modelagem das fases do *bot* na área de identificação e prevenção de atividades maliciosas. Também contribui com a produção de *traces* de *bot* simulados pertencentes à arquitetura centralizada fazendo uso do protocolo `HTTP` e mecanismo *pull* de requisições. Essas fases são as que produzem menos volume de tráfego se comparadas com as fases de ataque e propagação. As fases identificadas e modeladas represetam as fases de iniciais do ciclo de vida do *bot* e de manutenção de sua conectividade com a *botnet*.

Para isso, foi construído um simulador de *traces* de *bot* que é configurável, o que permite a parametrização de *bots* de diversos tipos de botnets C&C. As fases modeladas do *bot* estão contidas em seu *trace*, o que permite a identificação do comportamental do *bot* em relação ao seu C&C.

O arquivo de configuração do simulador possui trinta e três parâmetros o que permite uma combinação considerável de configurações para o *bot*. Além disso, diversos métodos são abstratos, o que permite a extensibilidade do simulador. A implementação dos métodos do cenário de sucesso pode ser reutilizada. Os cenários de exceção de execução da fase ou conexão ao servidor podem ser implementados para a execução do comportamento desejado para o *bot*.

Os *traces* usados para validação tiveram seu tráfego corretamente classificado dentro das fases propostas. Todas as fases do *bot* foram refletidas e identificadas no *trace* gerado. Os critérios de temporização, repetição e sequenciamento foram respeitados. A validação foi feita corretamente e comprovou a qualidade dos *traces* gerados que podem ser utilizados em pesquisas tanto na área de identificação e prevenção de atividades maliciosas, quanto na área de detecção e defesa.

Uma vantagem deste simulador é o tráfego gerado misturado naturalmente com o tráfego da máquina que o está executando, conforme ocorre com um *bot* num cenário real. Também de acordo com este cenário, o simulador gera tráfego de *bot* continuamente. Para fins de filtragem e análise do tráfego gerado, o simulador está configurado para adicionar o campo "`Marca-do-bot`" no cabeçalho `HTTP`. Isso é uma marcação de tráfego a nível de aplicação. Futuramente, trabalhos de marcação em níveis mais baixos de conexão,

como camada de Trasporte ou de Redes, poderão ser desenvolvidos e combinados com este simulador. A utilidade dessa marca vai de encontro ao uso de outros protocolos de comunicação de *botnets*.

A parametrização do simulador o torna facilmente configurável. Pode-se ainda reutilizar os métodos concretizados, modificá-los ou reescrevê-los totalmente, de acordo com o intuito do pesquisador. No *trace*, são refletidas as fases modeladas do *bot*.

O acesso ao simulador construído está aberto para instituições e pesquisadores, devidamente autorizados, para colaboração com seu crescimento e compartilhamento de dados gerados. Os *traces* resultantes, futuramente, poderão estar disponíveis em repositórios para fins de estudos acadêmicos.

Na tabela 7.1, está destacada a comparação das características das ferramentas que geram *traces* de *bot* e o simulador proposto.

TAB. 7.1: Comparativo de características entre as ferramentas que geram *traces* de *bot* e o simulador

Característica	**SLINGbot (JACKSON, 2009)**	**Rubot (LEE, 2009)**	**Simulador de *Traces* de *Bot***
Protocolo	Vários	Vários	HTTP
Arquitetura & Ambiente	ambas (enfoque *push*) engloba *bot* e servidor ambiente controlado	ambas (enfoque `P2P`) engloba *bot* e servidor ambiente controlado	centralizada(*pull*) *bot* ambiente real
Acesso	restrito	público	público
Implementação	Python, conjunto de funcionalidades	Ruby, conjunto de funcionalidades	JAVA, conjunto de atividades (Fases)

7.1 TRABALHOS FUTUROS

A simulação permite a coleta de tráfego de vários pontos da rede ao mesmo tempo e pode ser usada para simular uma *botnet*. A combinação do uso da simulação baseada na modelagem apresentada em conjunto com a inteligência da mudança constante dos nomes e `IPs` dos servidores C&C, com a utilização de técnicas de `Fast-Flux`[12], enriquece o *trace* a ser gerado. A técnica mais recente de `Bootkit`[13] permite a mundança de nomes de

[12]Técnica utilizada pela *botnet* para a mudança constante dos endereços `IPs` dos domínios dos servidores C&Cs, de acordo com (GU, 2008b)

[13]`Bootkit` em http://www.securelist.com/en/analysis/204792044/Bootkit_the_challenge_of_2008

domínio para o mesmo IP. Ela se assemelha as técnicas de Fast-Flux e de Domain Flux (STONE-GROSS, 2009). A implementação de DGAs pode ser feita nos pontos extensíveis do simulador onde é recuperado o nome do domínio, antes do envio da requisição. Tal fato constitui um campo aberto para estudos na área da gerência dos servidores C&C.

Apenas conexões que partem do *bot* para o C&C estão sendo abordadas. Aberturas de *backdoors* foram ignoradas por motivo de se tratar mais de atividades pertinentes às fases de propagação e ataque. Em trabalhos futuros, a implementação de *backdoors* nestas fases é uma contribuição a ser considerada.

A mudança de papel do *bot* na *botnet*, assim como o envio do IP da rede externa do *bot*, podem ser estudados a fim de se adicionar mais um nível de inteligência a este simulador. A implementação feita na linguagem JAVA permite que inúmeras bibliotecas possam ser utilizadas e faz com que o simulador seja portável. Esta é uma linguagem de programação bastante conhecida na atualidade, o que facilita a manutenibilidade do simulador por parte de vários pesquisadores e instituições.

Podem ser feitas melhorias na forma da leitura do arquivo de configuração do simulador com a utilização de outros formatos de extensão de arquivo, como o ".xml", por exemplo. O mesmo pode ser feito para os arquivos que contém os endereços dos servidores.

A modelagem produziu uma simulação configurável que possui extensibilidade, podendo ser refinada para a inclusão de outros protocolos de comunicação, mecanismo *push*, serviço distribuído de DNS, criptografia e autenticação do canal de comunicação, criptografia da requisição e outras atividades das fases de propagação e ataque. Também poderá ser realizado o refinamento das fases descritas neste trabalho.

São vários os pontos de extensibilidade do simulador. A combinação das fases apresentadas com as fases a serem modeladas (propagação e ataque) aumentam as funcionalidades do *bot* e o aproximam mais do cenário real.

8 REFERÊNCIAS BIBLIOGRÁFICAS

ABT, S. **Extending a framework for botnet simulation**, 2013a. URL `http://www.dasec.h-da.de/staff/abt-sebastian/botnet-simulation-framework`. Acessado em: 09/10/2013.

ABT, S. ***Missing Data Problem in Cyber Security Research***. Tese de Doutorado, Biometrics and Internet Security Research Group Hochschule Darmstadt, Darmstadt, Germany, 2013b.

AJELLI, M., CIGNO, R. L. e MONTRESOR, A. **Modeling Botnets and Epidemic Malware**. Em *Communications (ICC), 2010 IEEE International Conference on, Pages 1 - 5*, 2010. URL `http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=5502265`.

ARLITT, M. e WILLIAMSON, C. **A synthetic workload model for internet mosaic traffic**. *Proceedings of the 1995 Summer Computer Simulation Conference*, págs. 852–857, 1995.

BABI, C. Y. C. D. e SONG, E. C. R. S. D. **Inference and Analysis of Formal Models of Botnet Command and Control Protocols**. Em *CCS 10 Proceedings of the 17th ACM conference on Computer and communications security Pages 426-439*, 2010. URL `http://dl.acm.org/citation.cfm?id=1866355`.

BINSALLEEH, H., ORMEROD, T., BOUKHTOUTA, A., SINHA, P., YOUSSEF, A., DEBBABI, M. e WANG, L. **On the Analysis of the Zeus Botnet Crimeware Toolkit**. Em *Eighth Annual International Conference on Privacy, Security and Trust*, 2010.

CERON, J. M., GRANVILLE, L. Z. e TAROUCO, L. M. R. **Uma arquitetura baseada em assinaturas para mitigação de botnets**. Em *X Simpósio Brasileiro em Segurança da Informação e de Sistemas Computacionais (SBSeg), pages 105-118*, 2010.

CHOI, H., LEE, H. e KIM, H. **BotGAD: detecting botnets by capturing group activities in network traffic**. Em *Fourth International ICST Conference on COMmunication System softWAre and middlewaRE, COMSWARE 09, ACM, New York, NY, USA*, 2009.

CHOI, H. K. e LIMB, J. O. **A behavioral model of web traffic**. *ICNP 99: Proceedings of the Seventh Annual International Conference on Network Protocols, IEEE Computer Society*, págs. 327–334, 1999.

COCHRAN, W. G. **The chi-square test of goodness of fit**. *Annals of Mathematical Statistics*, 23:315–345, 1952.

COOKE, E., JAHANIANA, F. e MCPHERSON, D. **The zombie roundup: understanding, detecting, and disrupting botnets**. Em *Proceedings of the Steps to Reducing Unwanted Traffic on the Internet on Steps to Reducing Unwanted Traffic on the Internet Workshop, USENIX Association, Berkeley, CA, USA*, págs. p. 6–6, 2005.

DAGON, D., ZOU, C. e LEE, W. **Modeling Botnet Propagation Using Time Zones**. Em *13 th Network and Distributed System Security Symposium NDSS*, 2006. URL `http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.128.8689`.

DAGOSTINO, R. B. e STEPHENS, M. A. ***Goodness-of-fit Techniques***. New York, 1986.

ESTÉVEZ-TAPIADOR, J. M., GARCÍA-TEODORO, P. e DÍAZ-VERDEJO, J. E. **Measuring normality in HTTP traffic for anomaly-based intrusion detection**. *Computer Networks*, 45(2):175–193, 2004. ISSN 1389-1286.

FEILY, M., SHAHRESTANI, A. e RAMADASS, R. **A Survey of Botnet and Botnet Detection**. *Third International Conference on Emerging Security Information, Systems and Technologies*, 2009.

FOSSI, M., EGAN, G., HALEY, K., JOHNSON, E., MACK, T., ADAMS, T., BLACKBIRD, J., LOW, M. K., MAZUREK, D., MCKINNEY, D. e WOOD, P. **Symantec Internet Security Threat Report - Trends for 2010**. Technical report, Symantec, 2011. URL `https://www4.symantec.com/mktginfo/downloads/21182883_GA_REPORT_ISTR_Main-Report_04-11_HI-RES.pdf`. Acessado em: 03/11/2013.

FREIRE, E. P., ZIVIANI, A. e SALLES, R. M. **On Metrics to Distinguish Skype flows from HTTP traffic**. *Journal of Network and Systems Management*, 17:53–72, 2009.

GONCHAROV, M. **Russian Underground 101**, 2012. URL `http://www.trendmicro.com/cloud-content/us/pdfs/security-intelligence/white-papers/wp-russian-underground-101.pdf`. Trend Micro Incorporated Research Paper. Acessado em: 27/09/2013.

GU, G., PORRAS, P., YEGNESWARAN, V., FONG, M. e LEE, W. **BotHunter: detecting malware infection through IDS-driven dialog correlation**. Em *16th USENIX Security Symposium on USENIX Security Symposium, USENIX Association, Berkeley, CA, USA, 2007, pages 12-1–12-16.*, 2007.

GU, G., ZHANG, J. e LEE, W. **BotSniffer: Detecting Botnet Command and Control Channels in Network Traffic**. Em *15th Annual Network and Distributed System Security Symposium, The Internet Society (ISOC)*, 2008a.

GU, G., PERDISCI, R., ZHANG, J. e LEE, W. **BotMiner: Clustering Analysis of Network Traffic for Protocol- and Structure-Independent Botnet Detection**, 2008b.

HARLEY, D. **SODDImy and the Trojan Defence**. *4th International Conference on Cybercrime Forensics Education and Training (CFET)*, 2010.

JACKSON, A. W., LAPSLEY, D., JONES, C., ZATKO, M., GOLUBITSKY, C. e STRAYER, W. T. **SLINGbot: A System for Live Investigation of Next Generation Botnets**, 2009.

LEE, C. P. ***Framework for botnet emulation and analysis***. Tese de Doutorado, School of Electrical and Computer Engineering in Georgia Institute of Technology, 2009.

LILLARD, T. e GARRISON, C. P. ***Digital Forensics for Network, Internet, and Cloud Computing: A Forensic Evidence Guide for Moving Targets and Data***. Syngress, 2010.

LU, T.-T., LIAO, H.-Y. e CHEN, M.-F. **An Advanced Hybrid P2P Botnet 2.0**. Em *ICEIS (3)*, págs. 273–276, 2011.

MAH, B. A. **An empirical model of HTTP network traffic**. Em *Proceedings of 16th Joint Conference of the IEEE Computer and Communications Societies (INFOCOM 97)*, 1997. ISBN 0-8186-7780-5.

MASSEY, F. J. J. **The Kolmogorov-Smirnov test of goodness of fit**. *American Statistical Association*, 46:68–78, 1951.

MATROSOV, A., RODIONOV, E. e HARLEY, D. **TDSS part 2: Ifs and Bots**, 2011. URL `http://resources.infosecinstitute.com/tdss4-part-2/`.

MCPHERSON, D. **Botnet c&c quandry: Infiltrate or extirpate?** Technical report, Arbor Networks, 2007. URL `http://ddos.arbornetworks.com/2007/03/botnet-cc-quandry-infiltrate-or-extirpate/`.

NETWORKS, A. **Anatomy of a Botnet**, 2012. URL `http://www.arbornetworks.com/docman-component/doc_download/494-anatomy-of-a-botnet-white-paper`. Acessado em: 12/04/2013.

PORRAS, P., HASSEN, S. e YEGNESWARAN, V. **A Foray into Conficker's Logic and Rendezvous Points**. Em *Proceedings of the 2Nd USENIX Conference on Large-scale Exploits and Emergent Threats: Botnets, Spyware, Worms, and More*, LEET'09, págs. 7–7, Berkeley, CA, USA, 2009. USENIX Association. URL `http://dl.acm.org/citation.cfm?id=1855676.1855683`.

RICCARDI, M., PIETRO, R. D., PALANQUES, M. e VILA, J. A. **Titans' Revenge: Detecting Zeus via Its Own Flaws**. *Comput. Netw.*, 57(2):422–435, fevereiro 2013. ISSN 1389-1286. URL `http://dx.doi.org/10.1016/j.comnet.2012.06.023`.

RODIONOV, E., MATROSOV, A. e VOLKOV, D. **Hodprot: Hot to Bot**, 2011. URL `http://go.eset.com/us/resources/white-papers/Hodprot-Report.pdf`. Acessado em: 28/05/2013.

RSA. **RSA 2012 Cybercrime Trends Report**. Technical report, RSA, 2012.

SADHAN, B. A., MOURA, J. M. F., LAPSLEY, D., JONES, C. e STRAYER, W. T. **Detecting botnets using command and control traffic**. Em *Eighth IEEE International Symposium on Network Computing and Applications*, 2009. URL http://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=05190367.

SILVA, R. M. P. e SALLES, R. M. **Uma Estratégia para Detecção Online de Bots P2P**. *Workshop de Redes P2P, Dinâmicas, Sociais e Orientadas a Conteúdo (WP2P+/SBRC 2013)*, 1:1–6, 2013.

SILVA, S. S., SILVA, R. M., PINTO, R. C. e SALLES, R. M. **Botnets: A survey**. *Computer Networks*, 2012. ISSN 1389-1286. URL http://www.sciencedirect.com/science/article/pii/S1389128612003568.

SNORT. **Zeus Trojan Analysis**, 2010. URL http://labs.snort.org/papers/zeus.html. Sourcefire Vulnerability Research Team (VRT) Labs. Acessado em:13/11/2012.

STANKOVIC, S. e SIMIC, D. **Defense Strategies Against Modern Botnets**. *International Journal of Computer Science and Information Security (IJCSIS)*, 2(1), 2009.

STONE-GROSS, B. **The Lifecycle of Peer-to-Peer (Gameover) ZeuS**, July 2012. URL http://www.secureworks.com/cyber-threat-intelligence/threats/The_Lifecycle_of_Peer_to_Peer_Gameover_ZeuS/. Acessado em: 17/07/2013.

STONE-GROSS, B., COVA, M., CAVALLARO, L., GILBERT, B., SZYDLOWSKI, M., KEMMERER, R., KRUEGEL, C. e VIGNA, G. **Your botnet is my botnet: analysis of a botnet takeover**. Em *Proceedings of the 16th ACM conference on Computer and communications security*, CCS09, págs. 635–647, New York, NY, USA, 2009. ACM. ISBN 978-1-60558-894-0. URL http://doi.acm.org/10.1145/1653662.1653738.

STOVER, S., DITTRICH, D., HERNANDEZ, J. e DIETRICH, S. **Analysis of the Storm and Nugache trojans: P2P is here**. 2007. URL https://www.usenix.org/publications/login/december-2007-volume-32-number-6/analysis-storm-and-nugache-trojans-p2p-here.

STRAYER, W. T., LAPSLEY, D. E., WALSH, R. e LIVADAS, C. **Botnet Detection Based on Network Behavior**. Em LEE, W., WANG, C. E DAGON, D., editores, *Botnet Detection*, volume 36 of *Advances in Information Security*, págs. 1–24. Springer, 2008. ISBN 978-0-387-68768-1. URL http://dblp.uni-trier.de/db/series/ais/ais36.html.

STURGEON, W. **Net pioneer predicts overwhelming botnet surge**. Technical report, CNET News.com, 2007. URL http://news.cnet.com/2100-7348_3-6154221.html. Acessado em: 23/09/2013.

SYSFORENSICS. **Zeus v2 Malware Analysis Part I**, 03 2012. URL http://sysforensics.org/2012/03/zeus-v2-malware-analysis-part-i.html. Acessado em: 18/12/2013.

THOMPSON, T. P. **sandbox**, Setembro 2005. URL http://searchsecurity.techtarget.com/definition/sandbox. acessado em 20/01/2013.

TRENDMICRO. **Taxonomy of Botnet Threats**. Technical report, November 2006. URL http://www.cs.ucsb.edu/ kemm/courses/cs595G/TM06.pdf. Acessado em: 21/10/2013.

TYAGI, A. K. e AGHILA, G. **A Wide Scale Survey on Botnet**. *International Journal of Computer Applications (0975 - 8887)*, Volume 34-Number 9, 2011.

WANG, B., LI, Z., LI, D., LIU, F. e CHEN, H. **Modeling Connections Behavior for Web-Based Bots Detection**. *2nd International Conference on e-Business and Information System Security (EBISS)*, págs. 1 – 4, 2010.

WANG, P., SPARKS, S. e ZOU, C. C. **An advanced hybrid peer-to-peer botnet**. Em *Proceedings of the first conference on First Workshop on Hot Topics in Understanding Botnets*, HotBots'07, págs. 2–2, Berkeley, CA, USA, 2007. USENIX Association.

WANG, Y., WEN, S., ZHOU, W., ZHOU, W. e XIANG, Y. **The Probability Model of Peer-to-Peer Botnet Propagation**. Em *ICA3PP - Proceedings of the 11th international conference on Algorithms and architectures for parallel processing - Volume Part I*, págs. Pages 470–480, 2011a.

WANG, Y., ZHANG, Z., YAO, D. e GUO, B. Q. L. **Inferring Protocol State Machine from Network Traces: A Probabilistic Approach**. Em *ACNS11 Proceedings of the 9th international conference on Applied cryptography and network security Pages 1-18*, 2011b. URL http://dl.acm.org/citation.cfm?id=2025970.

WANGA, P., ASLAM, B. e ZOU, C. C. ***Peer-to-Peer Botnets***, chapter 18, págs. 335–350. Springer Press, 2010.

WILLEMS, C., HOLZ, T. e FREILING, F. **Toward Automated Dynamic Malware Analysis Using CWSandbox**. *IEEE SECURITY & PRIVACY*, págs. 32–39, março/abril 2007.

XIE, Y., YU, F., ACHAN, K., PANIGRAHY, R. e HULTEN, G. **Spamming Botnets: Signatures and Characteristics**. Em *In SIGCOMM*, 2008. URL http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.155.4526.

XIN-LIANG, W., LU-YING, C., FANG, L. e ZHEN-MING, L. **Analysis and Modeling of the Botnet Propagation Characteristics**. págs. 1 – 4, 2010. URL http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=5601301.

YONG, W., TEFERA, S. H. e BESHAH, Y. K. **Understanding Botnet: From Mathematical Modelling to Integrated Detection and Mitigation Framework**. Em *13th ACIS International Conference on Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing*, 2012.

YOUNG, I. T. **Proof without prejudice: use of the Kolmogorov-Smirnov test for the analysis of histograms from flow systems and other sources**. *Journal of Histochemistry and Cytochemistry*, 25(7):935–941, July 1977.

ZEIDANLOO, H. R. e MANAF, A. A. **Botnet Command and Control Mechanisms**. Em *Proceedings of the 2009 Second International Conference on Computer and Electrical Engineering - Volume 01*, ICCEE '09, págs. 564–568, Washington, DC, USA, 2009. IEEE Computer Society. ISBN 978-0-7695-3925-6. URL `http://dx.doi.org/10.1109/ICCEE.2009.151`.

ZEIDANLOO, H. R. e MANAF, A. B. A. **Botnet Detection by Monitoring Similar Communication Patterns**. *CoRR*, abs/1004.1232, 2010. URL `http://dblp.uni-trier.de/db/journals/corr/corr1004.html`.

ZHU, Z., LU, G., CHEN, Y., FU, Z. J., ROBERTS, P. e HAN, K. **Botnet research survey**. *Computer Software and Applications*, 32nd Annual IEEE International:967–972, 2008.

ZOU, C. C. e CUNNINGHAM, R. **Honeypot-aware advanced botnet construction and maintenance**. Em *International Conference on Dependable Systems and Networks (DSN06), pages 199-208*, 2006.

9 APÊNDICES

9.1 APÊNDICE 1: DIÁLOGOS DAS FASES CONFIGURAÇÃO, ATUALIZAÇÃO E KEEP-ALIVE

Este Anexo apresenta os seguintes códigos referentes aos diálogos das fases Configuração, Atualização e Keep-Alive, respectivamente. As transações referentes a estas fases estão representadas na figura 4.8.

Os códigos descrevem a estrutura geral de como cada fase é executada. Nesta estrutura, se encontra a inteligência de cada fase. O controle da lógica de sequenciamento e repetição entre as fases foi feito com o uso de *threads* e temporizadores.

```
FaseConfiguração(){
 SE "fase possui arquivo de configuração" ENTÃO{
      ENQUANTO "número de mensagens enviadas" <
          "número total de mensagens a serem enviadas" FAZER{
           recuperarServidor();
           SE "servidor deve ser checado" ENTÃO{
                checarServidor();
               SE "servidor não checado" E "conexão é persistente" ENTÃO{
                    tratarExcecaoChecagemServidor();
                   SE "tratamento de exceção mal sucedido" ENTÃO{
                        abortarFaseConfiguração();
                   }
               }
          }
          montarMensagem();
          enviarRequisição();
          SE "dupla requisição" ENTÃO{
               enviarDuplaRequisição();
          }
          SE "envio de requisição mal sucedido" ENTÃO{
               SE "conexão é persistente" ENTÃO{
                   tratarErroConfiguração();
                   SE "tratamento de erro mal sucedido" ENTÃO
                     { abortarFaseConfiguração(); }
               }
          }
      }
 } SENÃO{
      SE "dupla requisição" ENTÃO
         { enviarDuplaRequisição(); }
   }
 retornar resultado da Fase Configuração;
}
```

```
FaseAtualização(){
 SE "existe temporização para Keep-Alive" ENTÃO
     { Cancelar os temporizadores de Keep-Alive; }
 ENQUANTO "num msgs enviadas" < "num total msgs para enviar"  FAZER{
     recuperarServidor();
     SE "servidor deve ser checado" ENTÃO{
         checarServidor();
         SE "servidor não checado" E "conexão é persistente" ENTÃO{
             tratarExcecaoChecagemServidor();
             SE "tratamento de exceção mal sucedido" ENTÃO
                 { abortarFaseAtualização(); }
         }
     }
     montarMensagem();
     enviarRequisição();
     SE "envio de requisição mal sucedido" ENTÃO{
         SE "conexão é persistente" ENTÃO{
             tratarErroAtualização();
             SE "tratamento de erro mal sucedido" ENTÃO
               { abortarFaseAtualização(); }
         }
     }
 }
 SE "existe fase Configuração" E "atualiza arquivo de configuração"
    E "primeira execução da fase Atualização" ENTÃO{
     adicionarArqConfigEmListaArqsAtualização();
     incrementarNumeroArqsParaSeremAtualizados(); }
 SE "existe fase Keep-Alive" E "método GET" ENTÃO
   { dispararTemporizadorKeep-AliveMetodoGET(); }
 SE "existe fase Keep-Alive" E "método POST" ENTÃO
   { dispararTemporizadorKeep-AliveMetodoPOST(); }
  retornar resultado da Fase Atualização;
}
```

```
FaseKeep-Alive(){
 recuperarServidor();
 SE "servidor deve ser checado" ENTÃO{
      checarServidor();
      SE "servidor não checado" E "conexão é persistente" ENTÃO{
           tratarExcecaoChecagemServidor();
      }
  }
     montarMensagem();
     enviarRequisição();
     SE "envio de requisição mal sucedido"
        E "conexão é persistente" ENTÃO{
             tratarErroKeep-Alive();
     }
 disparar TemporizadorKeep-Alive();
}
```

9.2 APÊNDICE 2: DADOS COLETADOS DOS *TRACES* ORIGINAL E GERADO DOS EXPERIMENTOS

TAB. 9.1: Dados coletados dos *traces* original (Or) e gerado (Gr) para os três parâmetros do experimento 1 - ordem crescente

Intervalo Or.	**Intervalo Gr**	**Tam. Req. Or**	**Tam. Req. Gr**	**Tam. Resp. Or**	**Tam. Rsp. Gr**
0,00064	0,026275	134	142	252	469
0,536318	1,354134	135	150	252	469
0,994317	4,144039	307	307	469	469
5,512371	4,24808	307	307	469	469
5,514471	4,667895	307	307	469	469
5,532358	4,864982	307	307	469	469
5,546322	5,010306	307	307	469	469
5,610387	5,216037	307	307	469	469
5,701735	5,683914	307	307	469	469
5,7368	5,6951893	307	307	469	469
5,777386	5,872939	307	307	469	469
15,362768	8,66923	307	307	469	469
30,548092	31,87737	459	459	4083	4040
–	–	510	510	2032534	1154192

TAB. 9.2: Dados coletados dos *traces* original (Or) e gerado (Gr) para os três parâmetros do experimento 2 - ordem crescente

Intervalo Or.	**Intervalo Gr**	**Tam. Req. Or**	**Tam. Req. Gr**	**Tam. Resp. Or**	**Tam. Rsp. Gr**
0,003844	0,042329	164	229	236	321
0,495076	0,125516	165	260	236	321
1,369822	0,183384	370	355	285	321
30,163115	30,299349	458	483	20315	22001
–	–	542	483	35511	37447

TAB. 9.3: Dados coletados dos *traces* original (Or) e gerado (Gr) para os três parâmetros do experimento 3 - ordem crescente

Intervalo Or.	**Intervalo Gr**	**Tam. Req. Or**	**Tam. Req. Gr**	**Tam. Resp. Or**	**Tam. Rsp. Gr**
0,00036	0,013633	167	187	376	321
0,493648	0,099286	172	226	381	321
1,227918	0,200838	175	275	385	321
3,101195	1,030316	371	419	488	321
16,13843	2,216787	371	419	119675	127286
30,144719	30,962016	473	533	185512	163114
–	–	557	533	593886	588004

9.3 APÊNDICE 3: CONFIGURAÇÕES DAS MÁQUINAS DO EXPERIMENTO DA *BOTNET*

A seguir, serão listadas as configurações das máquinas físicas e virtuais do experimento do funcionamento do conjunto de *bots*.

Máquinas físicas:

- **Servidor:** Sistema operacional `Windows 7 32 bits Professional SP1`. Servidor *Web* `Wamp Server` 32 *bits* versão 2.4 que engloba o `Apache` (versão 2.4.4), o `MySQL` (versão 5.6.12) e o `PHP`(versão 5.4.16).
- **Bots:** Sistema operacional Ubuntu versão 12.04 LTS (*Long Term Support*)
- **Todas as máquinas:** memória `RAM` de `4GB`, processador `Intel core i5 3.100 GHz` com 4 núcleos de processamento, placa de rede gigabit `Intel` (on board). Gerenciador de máquinas virtuais `KVM` (*Kernel Virtual Machine*).

Máquinas virtuais (*bots*):

- Sistema operacional Ubuntu versão 12.04 LTS (*Long Term Support*)
- JVM (Java *Virtual Machine*)
- Foram alocados, como limite máximo, os 4 núcleos de processamento e `3GB` de memória `RAM` para cada máquina virtual.

Tanto o sistema operacional das máquinas reais quanto o das virtuais foram configurados para nunca interromper suas atividades. Isso foi feito para garantir o funcionamento contínuo dos *bots* e do servidor *web*.

9.4 APÊNDICE 4: CONFIGURAÇÕES GERAIS DOS *BOTS*

Configurações dos *bots* baseadas no arquivo de configuração do simulador e nos arquivos de servidores para cada fase.

Todas as fases:

- Nenhum *payload* é criptografado.
- Sem checagem de servidor.
- Sem persistência na conexão.
- *Timeout* padrão.
- Sem suspensão da fase Atualização antes da fase Keep-Alive.

Fase Configuração:

- Método `GET` de requisição para apenas um arquivo.
- `URL`: `192.168.7.1/botnet/arq2MB.exe`
- Não realiza atualização do arquivo de configuração.
- Possui dupla requisição com o uso do método `POST`.
- Intervalo para dupla requisição após a requisição do arquivo de configuração: 30 segundos.
- `URL` da dupla requisição: `192.168.7.1/botnet/ka_post.txt`
- Não envia mensagem de sucesso após *download* efetuado.
- *Payload* com dados de tamanho `266 bytes`.

Fase Atualização:

- Método `GET` de requisição para apenas um arquivo.

- `URL`: `192.168.7.1/botnet/wget.exe`
- Envia mensagem de sucesso após *download* efetuado.
- *Payload* com dados de tamanho `26 bytes`.
- Intervalo para próxima execução desta fase: 120 segundos.

Fase Keep-Alive - Método `GET`:

- `URL`: `192.168.7.1/botnet/ka_get.html`
- Sem *payload* na `URL`.
- Intervalo para próxima execução desta fase: 30 segundos.

Fase Keep-Alive - Método `POST`:

- `URL`: `192.168.7.1/botnet/ka_post.txt`
- *Payload* com dados de tamanho `118 bytes`.
- Intervalo para próxima execução desta fase: 30 segundos.