MINISTÉRIO DA DEFESA
EXÉRCITO BRASILEIRO
DEPARTAMENTO DE CIÊNCIA E TECNOLOGIA
INSTITUTO MILITAR DE ENGENHARIA
Seção de Engenharia de Computação / SE8

DAVI CATUNDA MOURÃO
LUCAS SOUZA SAMPAIO NUNES
LUIZ FELIPE DE BARROS RODRIGUES

MODELAGEM E SIMULAÇÃO DE UMA BOTNET PARA GERAÇÃO DE *TRACES* DE *BOT* NA ARQUITETURA ZEUS P2P

Rio de Janeiro
2014

INSTITUTO MILITAR DE ENGENHARIA

DAVI CATUNDA MOURÃO
LUCAS SOUZA SAMPAIO NUNES
LUIZ FELIPE DE BARROS RODRIGUES

MODELAGEM E SIMULAÇÃO DE UMA BOTNET PARA A GERAÇÃO DE *TRACES* DE *BOT* NA ARQUITETURA ZEUS P2P

Projeto de fim de curso apresentado ao Curso de Graduação de Engenharia de Computação como requisito parcial para a obtenção do título de Engenheiro.

Orientador: Ronaldo Moreira Salles.

Rio de Janeiro
2014

INSTITUTO MILITAR DE ENGENHARIA

DAVI CATUNDA MOURÃO
LUCAS SOUZA SAMPAIO NUNES
LUIZ FELIPE DE BARROS RODRIGUES

MODELAGEM E SIMULAÇÃO DE UMA BOTNET PARA A GERAÇÃO DE *TRACES* DE *BOT* NA ARQUITETURA ZEUS P2P

Projeto de Fim de Curso apresentado ao Instituto Militar de Engenharia, como requisito para colação de grau no Curso de Engenharia de Computação.

Orientador: Ronaldo Moreira Salles

Aprovada em ____ de _____de 2014 pela seguinte Banca Examinadora:

_____

Ronaldo Moreira Salles, Ph.D., do IME– Presidente

_____

Anderson Fernandes Pereira dos Santos, D.Sc., do IME

_____

Ricardo Choren Noya, D.Sc., do IME

Rio de Janeiro

2014

AGRADECIMENTOS

Primeiramente gostaríamos de agradecer a Deus e seu divino poder, que nos fez ser trazidos a essa escola de excelência em Engenharia, o IME. Sendo um instituto de referência em todo o país, devido à qualidade do ensino que há nessa Casa ministrada há séculos.

Agradecemos às nossas Famílias por ser nossa base e que nos apoiou para que esse trabalho fosse realizado e finalizado. Todos os amigos que nos deram forças para conclusão desse Projeto Fim de Curso.

Uma lembrança em especial ao nosso orientador Professor Salles. Conduziu-nos ao êxito nesse trabalho, para que alcançássemos essa importante contribuição para a sociedade acadêmica, além de ser uma fase importante para a nossa formação como engenheiro. Professor esse que é exemplo de grande profissional por sua competência, sabedoria e comprometimento, tanto no âmbito militar como acadêmico.

Contribuíram também todos os profissionais e professores do Departamento de Engenharia de Sistemas (SE/8). Em especial aos membros da banca, os Professores Choren e Anderson, que deram suas opiniões e sugestões para que esse trabalho fosse concluído com excelência.

Muito Obrigado a Todos!

Davi Catunda, Lucas Sampaio, Luiz Felipe Rodrigues

SUMÁRIO

LISTA DE ILUSTRAÇÕES

LISTA DE SIGLAS

C&C	*Comando e Controle*
DGA	*Domain Generation Algorithm*
DoS	*Denial of Service*
DDoS	*Distributed Denial of Service*
IDS	*Intruders Detection System*
IRC	*Internet Relay Chat*
HIDS	*Host Intruders Detection System*
HTTP	*Hypertext Transfer Protocol*
NIDS	*Network Intruders Detection System*
P2P	*Peer to peer*
TCP	*Transmission Control Protocol*
UDP	*User Datagram Protocol*

RESUMO

O rápido avanço das *botnets* e sua grande ameaça à Internet motivaram o estudo de formas de detecção de ataques às redes. No entanto, as pesquisas nessa área ainda são relativamente novas, levando ao crescente interesse no estudo desse assunto. O presente trabalho tem por finalidade modelar e simular os *traces* de uma *botnet* de arquitetura Zeus P2P, formada por *bots* infectados por *malwares*.

Nesse trabalho são apresentados os conceitos básicos de uma *botnet*, sua definição, tipos de arquitetura. Além disso, há uma explanação sobre a Zeus P2P que abrange sua estrutura, assim como as formas de comunicação dessa *botnet*. Modelam-se as fases de interesse do ciclo de vida do *bot*, para assim simular a *botnet*, baseada na arquitetura da Zeus P2P, e obter os *traces*.

Palavras chave: *botnet*, modelar, simular, Zeus, *trace*.

ABSTRACT

The fast advance of the botnets and the fact that they became a serious threat to the Internet motivated the study of ways of the detection of such attacks in the network. However, the researches in this area are still relatively new, taking to the growing interest on the study over this issue. This study aims to model and simulate traces of a botnet of Zeus P2P architecture, formed by bots, infected by malwares.

In this study are presented the concepts of a botnet, its definition, types of architecture. Furthermore, there is an explanation of Zeus P2P covering its structure, as well as the forms of communication in that botnet. The layers of interest in the life cycle of a bot were modelled, to simulate the botnet, based on architecture of Zeus P2P, and get the traces.

Key words: botnet, model, simulate, Zeus, trace.

1 INTRODUÇÃO

O uso de milhares de máquinas “escravas” para iniciar ataques em massa contra empresas e governos está se tornando uma tendência perigosamente comum (PURI, 2003). O mais preocupante é que estes ataques são descentralizados e de difícil contenção, dificultando sua detecção e prevenção.

Para conseguir derrubar o serviço de um servidor, são realizadas múltiplas requisições ao servidor por meio de milhares de máquinas infectadas distribuídas no mundo, até que seja atingido o limite físico de resposta do servidor.

Ações como essas são articuladas através do uso de *botnets* que são redes formadas por computadores infectados por *malwares* que realizam funções predefinidas de maneira automatizada (TYAGI, 2011), (SILVA, 2012), (COOKE, 2005).

Além destas máquinas infectadas realizarem ataques direcionados à empresas e governos, elas também podem roubar informações sigilosas como contas, senhas e dados pessoais, sendo este um assunto de grande importância na atualidade e de crescente preocupação.

Há três tipos de *botnets* disponíveis para estudo, as centralizadas, descentralizadas e as híbridas, sendo está última formada pelas duas primeiras. A centralizada se caracteriza pela existência de um centro de Comando e Controle (C&C), que troca informações com os *bots* da rede para atualizações e comandos, a partir de ordens dadas pelo *botmaster*, um operador humano que controla a rede. Já na descentralizada fica descaracterizado o C&C, dificultando a descoberta do servidor e a consequente fonte dos comandos da *botnet*.

Pela presença do C&C, na arquitetura centralizada, a comunicação nessa *botnet* se torna vertical, havendo a troca de informações somente do C&C para o *bot* e vice-versa. Diferentemente na descentralizada, que não há essa verticalização e a comunicação é *bot* para *bot,* com o *botmaster*, inclusive. Essa comunicação deixa rastros de tráfego de rede, os *traces*, que servem de insumo para ferramentas de análise e detecção de *botnets* (PACHECO, 2014).

1.1 OBJETIVO

O objetivo deste trabalho é desenvolver uma modelagem do comportamento da rede maliciosa, que abrange as fases do ciclo de vida de um *bot* assim como sua comunicação, e simular os *traces* de uma *botnet* conhecida como Zeus P2P, onde cada *bot* estaria em comunicação com outros *bots*.

Uma vez de posse dessa rede, será registrado todo o tráfego gerado por estes *bots*. Desenvolvendo desta forma uma ferramenta de referência para testar e comparar a eficiência de técnicas de detecção de tráfego malicioso.

1.2 MOTIVAÇÃO

As *botnets*, principalmente as descentralizadas, se tornaram grandes ameaças para a Internet. Segundo estimativas disponíveis em (STANIFORD, 2002), se um atacante tiver controle de um milhão de máquinas, poderá não existir defesa que seja capaz de frear um ataque de uma *botnet* desse porte. Um ataque dessa magnitude pode causar grandes danos à rede, podendo ser utilizado em guerras e ataques terroristas.

Diversos incidentes de segurança têm ocorrido por causa dessas pragas virtuais, o que tem motivado o estudo para a identificação e prevenção das *botnets*. Nosso estudo é acerca dos *traces*, rastros de tráfego na rede, dos *bots*, que serve de insumo para esse estudo na identificação e prevenção dessas redes.

No entanto, existe uma limitação da quantidade de *traces* com atividades de *botnets* com acesso público para fins de estudo na área. Isso dificulta bastante o desenvolvimento e a validação de novas técnicas de detecção (SILVA, 2012), (TYAGI, 2011). Os motivos recaem sobre a possibilidade da existência de informações pessoais e/ou sigilosas nesses *traces* (SILVA, 2012).

Visando preencher essa lacuna, esse trabalho modela o comportamento de uma *botnet* conhecida como Zeus P2P*,* a partir da estruturação da comunicação do *bot* e seu ciclo de vida, gerando, assim, o tráfego da rede.

1.3 METODOLOGIA

Para atingir o objetivo proposto, o trabalho foi dividido em 5 (cinco) etapas:

1. Compreender como os *bots* e *botnets* se comportam na arquitetura Zeus

P2P.

2. Modelar as fases do ciclo de vida de um *bot*.
3. Modelar a comunicação na *botnet*.
4. Montar uma infraestrutura capaz de atender os requisitos para simulação de uma *botnet*.
5. Gerar o código para a simulação de *traces* e obter o *trace* gerado pelos *bots*.

1.4 ORGANIZAÇÃO DA MONOGRAFIA

Este trabalho está organizado em 6 (seis) capítulos.

No segundo capítulo, realiza-se um estudo sobre *bots* e *botnets* contemplando o contexto no qual se enquadram, e os tipos de arquitetura desta rede.

No terceiro capítulo, estuda-se mais detalhadamente como a *botnet* Zeus P2P, assim como se dá sua estrutura e comunicação e tipos de *bots*.

No quarto capítulo, realiza-se o detalhamento da modelagem do ciclo de fases de um *bot* a serem usadas e da máquina de estados a ser utilizada futuramente no desenvolvimento do simulador de *bots*.

No quinto e sexto capítulo, faz-se o levantamento, a análise e a especificação de requisitos. Além disso, realiza-se o desenvolvimento do projeto para a geração do código e obtenção do *trace* em si.

No sétimo capítulo, realiza-se uma conclusão sobre o trabalho, falando sobre os resultados obtidos.

2 BOTNET

2.1 DEFINIÇÃO

Botnets são redes formadas por computadores "escravos" que foram infectados, chamados de *bots* (derivada da palavra inglesa *robot*), e são controladas por um ou mais atacantes chamados de *botmasters* (SILVA, 2012). Essa infecção se dá a partir do momento que é instalado na máquina vítima o *malware*, que são programas de cunho malicioso e fazem esses computadores realizarem, de forma automatizada, atividades controladas por um *botmaster* (CHOI, 2009).

Esses *malwares* são difundidos pela internet por auto-propagação com o intuito de aumentar o número de *bots* da rede. Com essa grande formação, as *botnets* são utilizadas na prática do *phishing*, que é obter dados pessoais das vítimas via e-mails, realizar ataques DDoS, cibercrimes e ciberataques em geral.

2.2 HISTÓRICO DAS BOTNETS

Inicialmente, as *botnets* foram originadas a partir da *Internet Relay Chat* (IRC), protocolo de comunicação voltado para a utilização em bate-papo e troca de arquivos (SILVA, 2012). Nesse período, o conceito de *bots* não era, ainda, estritamente ligado ao comportamento malicioso. Eles tinham a função de proteger canais, realizar as interações entre os canais, executar conversas, ou seja, automatizar, de múltiplas maneiras, tarefas relacionadas a *chats*, *online games*, *Multi-User Dungeon* (MUD), entre outros (HUDAK, 2006).

Então, os primeiros *bots* maliciosos apareceram e eram desenvolvidos usando a mesma ideia do *bot* não-malicioso, mas de maneira a automatizar tarefas maliciosas. Por exemplo, o primeiro *bot* IRC foi o *Eggdrop*, e seus primeiros *bots* maliciosos foram usados para atacar outros usuários IRC ou até outros servidores. É difícil identificar o momento em que as *botnets* se tornaram realidades, mas *Sub7* e *Pretty Park* – um *Trojan*, programa malicioso, e um *worm*, programa autorreplicante, semelhante a um vírus – são considerados *malwares* que contribuíram com a ascensão da *botnet* (BREWSTER, 2010). Estes foram identificados antes da virada do milênio e introduziram o conceito de um *host* infectado se conectando a um canal

IRC para receber comandos maliciosos.

Sequencialmente, as *botnets* iniciaram ataques DoS (negação de serviço) e DDoS (distribuído de negação de serviço) contra servidores. Os novos *bots* passaram a utilizar mecanismos mais heterogêneos com o *botmaster*, que, por sua vez, exploraram novos protocolos e novos tipos de ataques, tornando os *bots* mais robustos. Assim, eles podiam se propagar rapidamente, se esconder facilmente e, ao mesmo tempo, lançar ataques coordenados de grande porte. Os atacantes criaram maneiras mais simples de controlar os *bots*, a partir da utilização de protocolos, como P2P, HTTP, IRC, entre outros, ao invés de usar a arquitetura Comando e Controle, que será explicado mais à frente.

2.3 COMPONENTES DE UMA BOTNET

Para entender como uma *botnet* funciona, deve-se conhecer os seus componentes básicos, apesar das várias estruturas possíveis para uma *botnet*. São eles o *bot*, a máquina vítima, *botmaster* e o centro de Comando e Controle.

- *Bot* é uma máquina vítima com um *malware* instalado capaz de executar várias funções autônoma e automaticamente. *Bots* podem ser considerados legais ou ilegais, dependendo do seu uso. O *malware* pode ser instalado facilmente por acesso aos *sites* ou vírus da rede em que a máquina vítima está inserida. A *botnet*, por sua vez, é a rede formada pelo conjunto de todos os *bots*;
- Máquinas vítimas são os computadores que podem ser infectados por um *malware* e, geralmente, o alvo do ataque;
- *Botmaster* é a máquina que controla uma *botnet,* emitindo comandos aos *bots* para que estes realizem as atividades maliciosas;
- Centro de Comando e Controle (C&C) é a infraestrutura que serve para comandar os *bots*, necessitando de uma conexão entre o *bot* e o centro. Esse centro é comandado pelo *botmaster* que controla remotamente as ações que ele desejar para seus *bots* executarem.

2.4 TIPOS DE ARQUITETURA

A maneira que os *bots* formam uma *botnet*, através da infecção por *malwares*, a rede pode ser classificada de acordo com sua arquitetura específica e os modos em que opera. Podem ser classificadas em centralizada, descentralizadas e híbridas.

2.4.1 CENTRALIZADA

A arquitetura centralizada de uma *botnet* se aproxima bastante a uma arquitetura cliente-servidor em uma rede de computadores (SILVA, 2012). Neste tipo de arquitetura, todos os *bots* estabelecem conexão com um ou mais C&C que são responsáveis por enviar comandos ao *bots* e realizar atualização do *malware*.

Esta arquitetura possui diversas vantagens. A primeira é a baixa latência na comunicação, ou seja, um baixo tempo de resposta, pois um *bot* se comunica diretamente com um C&C. Segundo, tem-se o controle rígido da *botnet* pelo *botmaster*, pois ele possui controle sobre os C&C's, consequentemente sobre os *bots*. Por fim, outra vantagem seria o maior sincronismo na comunicação dado que todas as ações são coordenadas pelo *botmaster*, como ilustrado na **Figura 2.1**.

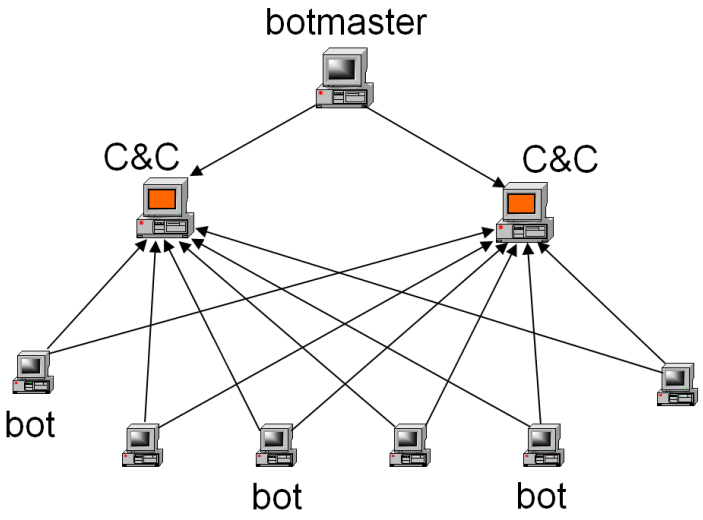


FIG 2.1. Arquitetura Centralizada (WANG, 2007)

Como desvantagem, pode-se citar a baixa resiliência, ou seja, caso um C&C seja detectado, a sua desarticulação impactaria diretamente na exposição da *botnet*. Outra desvantagem é o fato de que a captura de um *bot* pode comprometer toda a *botnet*, caso esse possua toda a lista de servidores C&C. Além disso, o comando de ataques sincronizados gera grande volume de pacotes enviados dos C&C's para os

bots.

Os protocolos mais utilizados em arquiteturas centralizadas são o *Internet Relay Chat* (IRC) e o *Hypertext Transfer Protocol* (HTTP). No caso dos *botnets* IRC, o *botmaster* cria vários canais IRC por onde os *bots* se comunicam. A vantagem do uso deste protocolo é a flexibilidade que é dada a *botnet*, pois o *botmaster* pode focar em um ataque na rede ou realizar vários ataques simultâneos. No entanto, existe uma desvantagem atrelada a essa vantagem, pois mesmo que seja um protocolo flexível, o protocolo IRC é raramente utilizado nos dias de hoje, tornando sua detecção mais fácil. Por isso, ultimamente, o protocolo HTTP tem sido utilizado, por ser mais difícil de ser detectado.

2.4.2 DESCENTRALIZADA (P2P)

Atualmente, as técnicas de detecção estão mais sofisticadas, exigindo um aumento na flexibilidade e na robustez dessas redes. Uma saída encontrada foi não utilizar um servidor C&C central, ou seja, descentralizar o comando da *botnet*, como é visto na **Figura 2.2***.* As *botnets* que possuem arquiteturas descentralizadas são mais difíceis de serem desarticuladas, pois a descoberta de diversos *bots* não acarreta, necessariamente, na falha da *botnet* por inteira porque não há C&C central para ser localizado e desativado (SILVA, 2012).

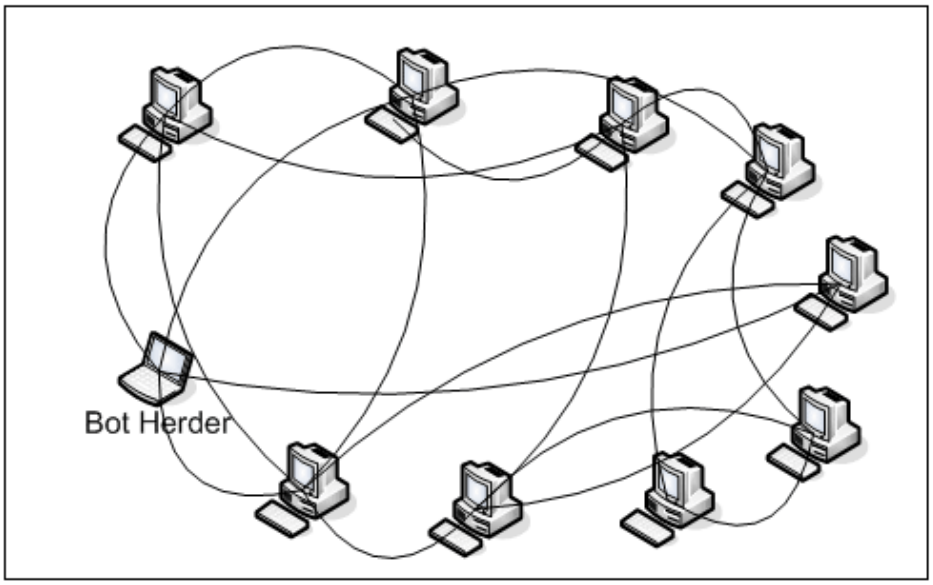


FIG 2.2. Arquitetura Descentralizada

Essas *botnets* são comumente baseadas numa variedade de protocolos P2P e

atuam como uma rede de sobreposição e podem ser classificadas como P2P estruturada, P2P não-estruturada e s*uperpeers*, que diferem quanto à maneira de construir a rede de sobreposição.

2.4.3 HÍBRIDA

A arquitetura híbrida possui características tanto da arquitetura centralizada quanto da descentralizada. Um exemplo bastante comum é a utilização de uma *botnet* com arquitetura P2P com *superpeers*, onde alguns *peers* atuam temporariamente e rotativamente como C&C.

Os *bots* pertencentes a um *botnet* P2P híbrida são classificados em *bots* clientes e *bots* "serventes", como é visto na **Figura 2.3**. Estes últimos podem atuar como *bots* clientes e *bots* servidores e possuem IP estático e são os únicos que podem ter seu endereço IP na lista de *peers* para serem conectados. Os *bots* clientes, por outro lado, possuem IP dinâmico e não aceitam conexões.

Esse é um tipo de arquitetura que consegue funcionar muito bem, pois os clientes só se conectam a *peers* servidores, pois só eles entram na lista de *peers.* Assim como acontece na arquitetura Centralizada, os clientes se conectam a todos os servidores presentes em suas listas de *peers* para atualizações.

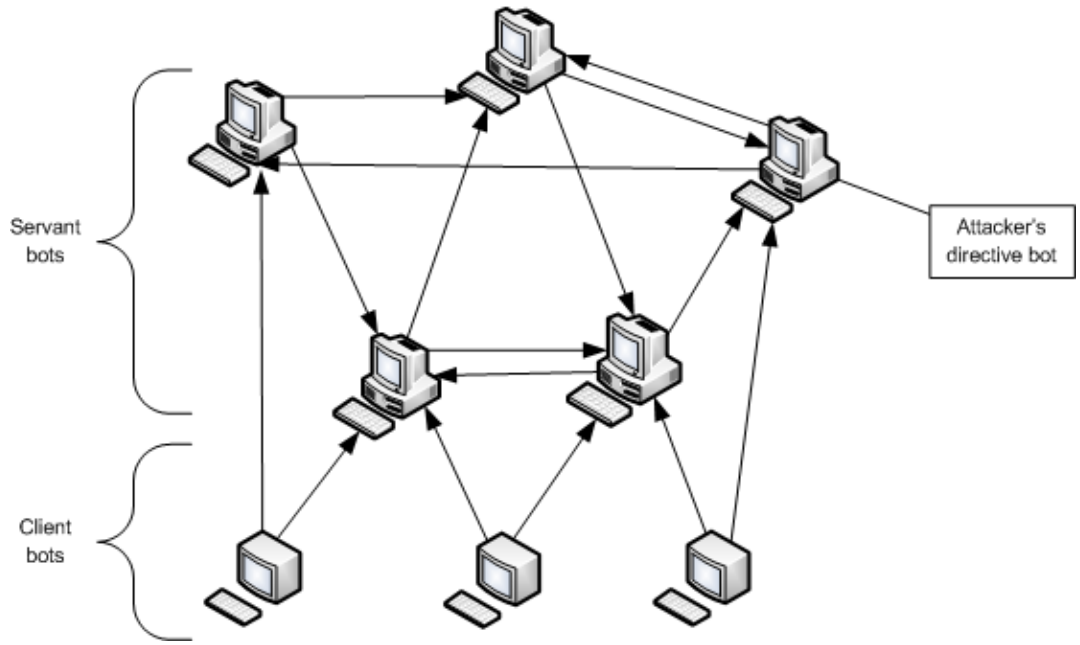


FIG 2.3. Arquitetura Híbrida

A rede que será utilizada como base para nossos estudos é a *botnet* Zeus P2P, que é híbrida e será mais detalhada no próximo capítulo.

3 ZEUS P2P

3.1 INTRODUÇÃO

Zeus é uma família de *trojans* que apareceu em 2007. As duas primeiras variantes de Zeus eram baseadas em servidores centralizados. Com o passar do tempo, esses servidores passaram a ser detectados e bloqueados. Para resistir a essas medidas, a segunda versão de Zeus foi direcionada a uma variante *peer-to-peer*.

Devido a sua grande popularização, a primeira versão de Zeus, que é baseada servidores Comando e Controle (C&C) centralizados, foi incessantemente investigada pela comunidade. Em 2011, no entanto, a segunda versão centralizada de Zeus modificou-se para uma variante *peer-to-peer (*P2P). A rede principal P2P é dividida em diversas *sub-botnets.* Enquanto a rede Zeus P2P é periodicamente atualizada, as *sub-botnets* são independentemente controladas para realizar variadas atividades maliciosas (ANDRIESSE, 2013).

O Zeus P2P realiza duas funções principais: os *bots* trocam atualizações de arquivos em binário e de configuração*,* onde comandos podem ser atualizados ou descartados, e trocam a lista de *proxy bots* e *proxy list.*

Outras diferenças são notadas comparando o Zeus P2P com a primeira versão do Zeus. Entre elas, está a utilização do mecanismo DGA que atua como *backup* para a geração de domínios. O *Domain Generation Algorithm* (DGA) é usado pelo *malware* para criar domínios objetivados a munir a *botnet* com atualizações ou comandos. Por exemplo, um *bot* infectado pode criar vários *sites* para entrar em contato com outros *bots* para enviar atualizações e comandos.

Ao longo deste capítulo, serão abordados detalhes acerca da topologia, dos protocolos de comunicação e a estrutura das mensagens da rede Zeus P2P.

3.2 TOPOLOGIA DA REDE

A rede Zeus é organizada em três camadas disjuntas, como mostrado na **Figura 3.1**. Na base da hierarquia está a camada P2P, que contém a maioria dos *bots*.

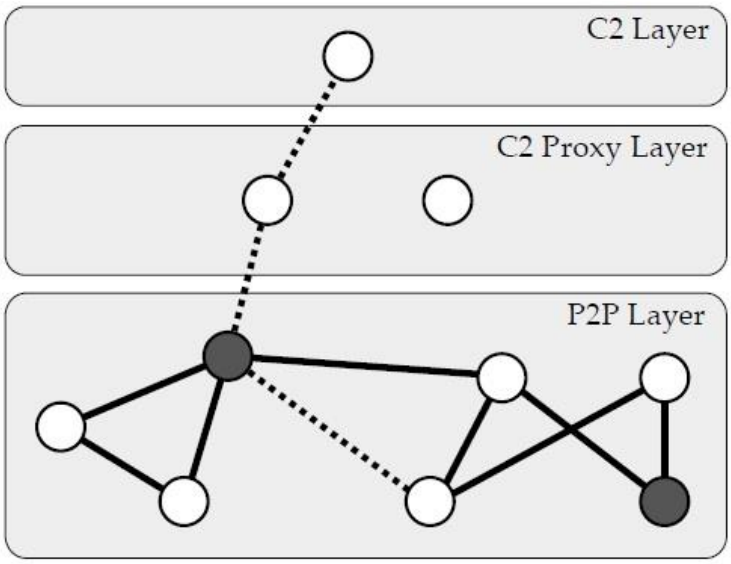


FIG 3.1. Arquitetura P2P Zeus (ANDRIESSE, 2013)

Periodicamente, um subconjunto destes *bots* muda de função e se torna um *proxy bot*, que está sombreado na **Figura 3.1**. Isto parece ser feito manualmente pelo *botmaster*, e é alcançado ao se enviar uma mensagem sinalizada anunciando este *proxy* para a rede. Os *proxies bots* são utilizados pelos *bots* comuns para atualizar seus comandos e para guardar dados roubados. Além das atividades de *proxy*, estes *bots* também se comportam como qualquer outro bot.

Uma importante variação do comportamento dos *proxies bots* foi introduzida em alguns tipos de *Botnets* Zeus em março de 2014. Estas variantes usam o DGA como principal canal C2 e revertem para *proxy bots* somente se o modo DGA não puder ser alcançado. Contudo, agora em maio de 2014 ainda se pode encontrar variantes que ainda usam o *proxy bots*, enquanto as outras usam como padrão o modo DGA. Ainda não está claro o que se tornará o padrão.

Os *proxy bots* agem como intermediários entre a camada P2P e uma camada mais alta, chamada de camada C2 de *proxies*. Esta camada contém vários servidores HTTP, os quais não são *bots*, que formam uma camada adicional entre os *proxies bots* e a verdadeira raiz da comunicação C2. Periodicamente os *proxies bots* interagem com a camada C2 com a finalidade de atualizar sua lista de comandos

maliciosos e para passar dados roubados coletados dos *bots* para cima na hierarquia.

Finalmente, no topo da hierarquia, está a camada C2, que é a raiz de todos os comandos e o destino de todos os dados roubados. Os comandos se propagam para baixo, da camada C2 atravessando a camada C2 de *proxies* chegando aos *proxies bots* aonde os *bots* comuns se atualizam. De forma similar os dados roubados são periodicamente propagados para cima até chegar à camada C2.

3.3 PROTOCOLOS E MENSAGENS

A grande inovação existente na variante P2P da Zeus é a comunicação por protocolos *peer-to-peer*, que tem como objetivo descentralizar o comando da *botnet*. A maior parte da comunicação realizada entre *bots* é baseada no protocolo UDP. As exceções são: comunicação entre *bots* e *proxies,* e troca de atualizações de arquivos em binário e de configuração que são baseadas no protocolo TCP.

As mensagens desta *botnet* variam em tamanho, mas tem no mínimo um cabeçalho de 44 *bytes*, já o restante é utilizado para armazenar o *payload* e o *padding* da mensagem. A seguir serão descritas algumas opções de mensagens possíveis para esta rede.

1. ***Version request (UDP)***: são usadas para requerer o número da versão dos arquivos binários e de configuração.
2. ***Version reply (UDP)*****:** É a resposta do *version request* e contém a versão dos arquivos binários e de configuração.
3. ***Peer list request (UDP)***: são usadas para abrir conexões com novos *peers* de outros *bots.*
4. ***Peer list reply (UDP)***: é a resposta para o *peer list request*, contendo 10 *peers* do *bot* responsivos que estão mais perto do *bot* que está requerendo esta lista.
5. ***Data request (TCP)*****:** são utilizadas para requerer um arquivo de configuração ou binário.
6. ***Data reply (TCP)*****:** é a resposta do *data request*, e é composta dos arquivos requeridos.

3.4 THREADS PASSIVOS E ATIVOS

Todo *bot* possui um thread passivo que fica a escuta de novas mensagens de outros *bots*. O *bot* Zeus recebendo um *request* deve lidar com ele da melhor forma possível, e tentar enviar de volta uma resposta apropriada.

O transmissor de qualquer *request* tratado corretamente é adicionado à lista de *peers* do *bot* receptor. Este é o principal mecanismo usado pelo *malware* Zeus para conhecer sua vizinhança, e é como os novos *bots* se introduzem à rede. Se o *bot* receptor tiver menos de 50 vizinhos, então ele adiciona o *bot* transmissor a sua lista de *peers*, entretanto se o *bot* já estiver na lista, suas informações de porta e de IP são atualizadas, visando acomodar *bots* com IPs dinâmicos e descartar IPs antigos. Caso já possua 50 *bots*, ele é adicionado a uma fila para ser adicionado em uma próxima verificação de inatividade na vizinhança (ANDRIESSE, 2013).

Antes de adicionar um *peer* à lista de *peers*, alguns testes são realizados. Primeiro, é verificado se a porta está entre 10000 e 30000. *Bots* NAT só podem entrar na lista de outro *bot* se eles escolherem uma porta no intervalo correto. Adicionalmente, somente um IP por cada subrede /20 pode estar presente na lista.

A maior parte das mensagens são *requests* de outros *peers*, com exceção dos anúncios dos *proxies peers*. Quando estes anúncios chegam, é feito uma verificação na sua assinatura, e se for correta, um campo armazena o número de ‘hops’ de vida da mensagem é decrementado e a mensagem é passada adiante para todos os seus vizinhos enquanto este campo ainda for positivo.

Além disso, novos *proxies* que passam na verificação podem ser adicionados à lista de *proxy bots*. Esta lista é similar a lista de *peers*, mas é mantida separadamente. Se uma entrada da lista de *proxy* é 100 minutos mais velha que o novo *proxy*, ela é sobrescrita com a nova, tendo a lista no máximo 10 entradas (ANDRIESSE, 2013).

Já o thread ativo se consiste de um ciclo que acontece a cada 30 minutos, aonde o *bot* entra em contato com cada um de seus *bots* vizinhos, requerendo suas versões de arquivos de configuração e binários, mantendo o *bot*, desta forma, atualizado. Além disso, também se aproveita esta etapa para verificar a conectividade dos *bots*. Se algum *bot* não responder ao primeiro *request*, mais 5 tentativas de contato serão feitas. Se mesmo assim o *bot* não responder, verifica-se

se o próprio *bot* que está enviando os *requests* está conectado à internet. Ao fim deste processo removem-se os bots não responsivos.

Finalmente, se a lista de *peers* contiver menos de 25 *bots* ativos, então o *bot* irá ativamente enviar *peer list requests* para seus *bots* vizinhos a fim de não ficar isolado, mas se mesmo assim não conseguir mais *bots*, então ele deve entrar no modo DGA.

4 MODELAGEM

Para a modelagem das fases do *bot* na arquitetura descentralizada, foi necessário pesquisar os critérios que regulam o sequenciamento de atividades nessa *botnet*. A partir dos critérios de sucessão de atividades, temporização e fluxo de comunicação, foram modeladas as fases do ciclo de vida do *bot*. Com o estudo de artigos científicos e relatórios técnicos citados posteriormente nas referências bibliográficas, foi possível estabelecer uma base de estudo para a análise das fases de identificação de atividades maliciosas. Assim, nesta seção, estão descritas as fases que compõem o ciclo de vida do *bot*.

A modelagem do comportamento geral do *bot* foi realizada a fim de se desenvolver o simulador. Foram adotados padrões para esta modelagem, como: a maioria das comunicações entre *bots* é feita via UDP, enquanto a comunicação entre *bots* e *proxy bots* e a troca de arquivos de atualização e configuração são feitas baseadas no protocolo TCP (ANDRIESSE,2013). Para manter a coerência da rede, a troca de *lista de peers* é feita pelo mecanismo *push*/*pull*.

4.1 CICLO DE VIDA DO BOT

Inicialmente foi feita uma máquina de estados com todas as fases do ciclo de vida do *bot*, construída partindo do estudo da área de *botnets*, considerando tanto a arquitetura descentralizada quanto a centralizada (PACHECO, 2014). As nove fases são descritas a seguir:

- Recém-Infecção: abertura de conexões com diversos *peers*, logo após a infecção da máquina;
- Configuração: requisições a *peers* vizinhos por arquivos diferentes do executável do *malware*;
- Identificação: requisição a um *peer* vizinho do IP do *bot* que se situa na *lista de peers*, que contém também suas portas e identificadores;
- Atualização: requisição a um *peer* com versão mais recente do *malware* por sua atualização. Esta é a primeira fase iterativa. Deste estado, podem-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: Todas as etapas da *botnet* tem como objetivo principal chegar neste ponto, aonde serão realizados ataques para obtenção de informações e para execução de mais ataques, como, por exemplo, *HTTP Injection*, *Selective screen scraping*, *Keylogging*, estabelecimento de uma *Back door*, *download* e execução de código remotamente, roubo de credencias (HTTP, FTP, pop3), acesso à VNC, DDoS, interceptar *HTTP forms*, *URL redirection*;

- *Report* do status do *bot*: Controle do status do *bot* no sequenciamento das fases, verificando respostas de sucesso e atualizações;

- Verificação de conectividade à rede (*Keep-Alive*): O *bot* envia requisições de tempos em tempos aos seus vizinhos, nessa etapa ele testa o status de sua conectividade e de seus vizinhos a rede. A periodicidade dos fluxos de comunicação neste estado é um traço marcante do *Keep-Alive*;

- Autopropagação: Fase em que ocorre a propagação do *malware* de diversas maneiras seja por *spams*, injeções de código, *keylogging* e outras infecções através de mídias removíveis; e

- Mudança de papel do *bot* na rede: Periodicamente, um subconjunto do conjunto de *bots* da *botnet* é designado como *proxy bot*, através de uma mensagem divulgada na rede para anunciar qual *bot* será um *proxy*. Os *proxies bots* são utilizados para buscar novos comandos e entregar informações extraídas da máquina infectada. Assim, periodicamente, essa *proxy list*, que contém a lista dos *proxies bots*, é alterada.

4.2 FASES DO MODELO DE CICLO DE VIDA DO BOT

Para que uma máquina infectada se torne um *bot* ativo e, assim, parte da *botnet*, a máquina deve passar por algumas fases. O conjunto dessas fases caracteriza o ciclo de vida de um *bot*, o qual pode variar de acordo com a estrutura da *botnet*. Essas fases podem ter nomes diferentes dependendo da literatura utilizada como pesquisa e da implementação, sendo, a arquitetura da *botnet* Zeus P2P, utilizada

como foco nesse estudo.

As fases modeladas foram as de Recém-Infecção, Configuração, Atualização e *Keep-Alive* por se tratarem de partes relacionadas à identificação de atividades maliciosas. Por motivos de simplificação, não será modelado a parte iterativa do modo DGA da fase de Recém-Infecção, sendo somente modelado o fluxo onde a conexão não falha e o *download* do arquivo de configuração é feito com sucesso, contudo será implementado uma interface extensível para futuras implementações. Desta forma as fases de Recém-Infecção e de Configuração somente são realizadas uma vez, não necessitando sua temporização. Um estudo sobre o comportamento temporal da rede pode ser encontrado em (STRAYER, 2008) e (SILVA, 2013).

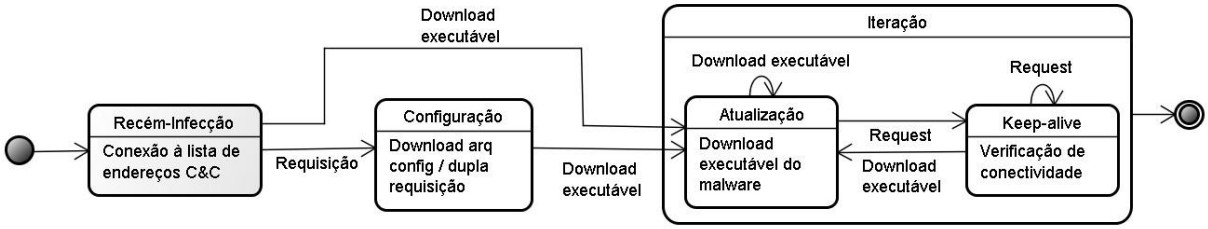


FIG 4.1. Máquina de Estados Modeladas do *Bot* (PACHECO, 2014)

Na **Figura 4.1** foram representadas as fases e como elas interagem entre si, aonde cada fase será mais aprofundada nos próximos tópicos e suas interações mais detalhada na **Figura 4.2** na qual cada interação entre estados é uma mensagem ou uma ação tomada pelo *bot* nas 4 fases descritas.

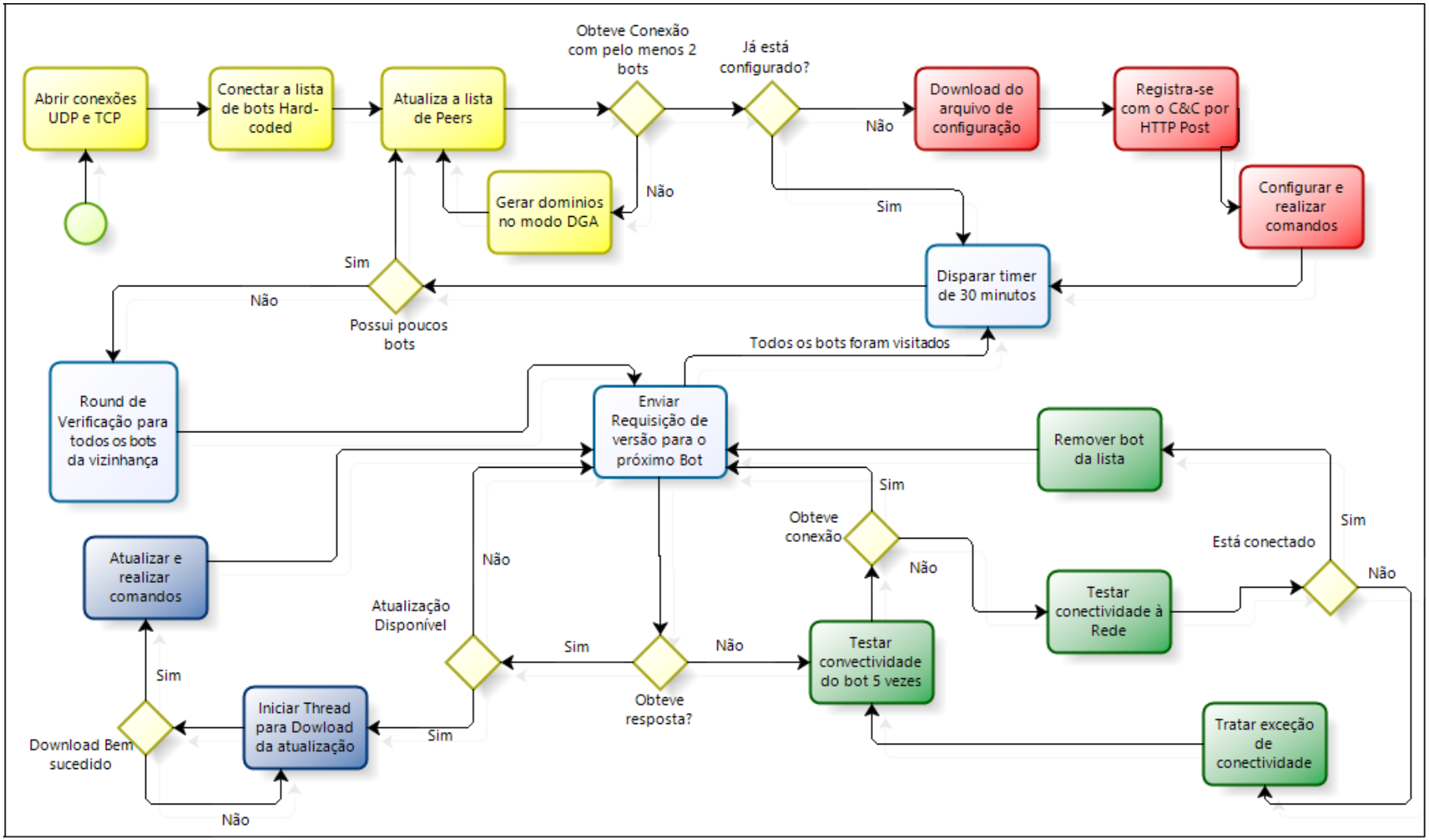


FIG 4.2. Diagrama das fases Identificadas em uma *botnet* P2P. (categorizadas por uma cor)

Recém-Infecção(Amarelo); Configuração(Vermelho); Atualização (Azul); Keep-Alive(Verde)

4.2.1 RECÉM-INFECÇÃO

Nesta fase, a máquina acaba de ser infectada, tornando-se um *bot* em potencial. Essa infecção pode ocorrer através de *downloads* de *malwares* sem solicitação, dispositivos móveis infectados, abertura de *spam*, anexos infectados em e-mails, entre outras maneiras (ZHU, 2008). Assim que a máquina obtém acesso à Internet o *malware* do Zeus P2P é executado e imediatamente abre diversas conexões com uma série de *peers hard-coded* no programa para requerer arquivos de configuração, *binary updates* e listas de *peers*. A lista de *peers* contém o endereço de IP, portas e identificadores únicos de por volta de 50 *bots* Zeus; as portas variam entre 10000 e 30000; os identificadores tem o tamanho de 20 *bytes* e são gerados no momento da infecção fazendo um *hash* (SHA-1) no nome do Computador e na ID do primeiro volume do *hard-drive* (ANDRIESSE, 2013). Esses identificadores são utilizados para manter uma informação de contato atualizado de *bots* com IPs dinâmicos.

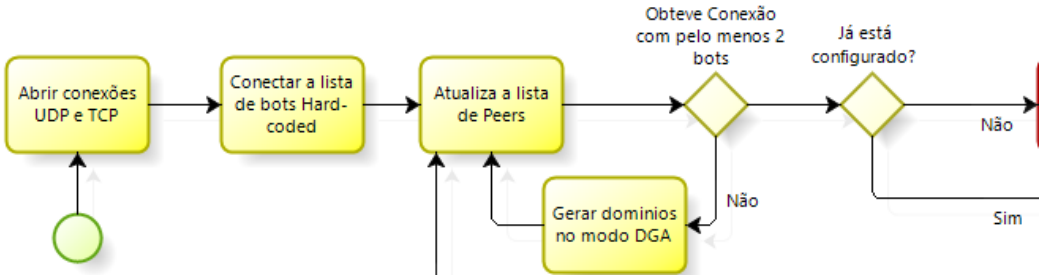


FIG 4.3. Diagrama de Transição – Recém-infecção.

Se os *peers hard-coded* não puderem ser alcançados, P2P Zeus entra no modo DGA utilizando a data corrente como *semente*. O DGA produz 1000 domínios pseudorrandômicos por dia. Os nomes dos domínios possuem entre 32 e 48 caracteres de comprimento com um dos seis domínios *top-level* (com, net, org, biz, info, e ru). Esses domínios proveem uma nova lista de *peers* possivelmente ativos na rede. Cada entrada de *peer* é armazenada como uma tupla composta de *Bot*, ID, endereço Ipv4, porta, endereço Ipv6, e porta. Zeus P2P suporta tanto endereços Ipv4 como Ipv6 (STONE-GROSS, 2012).

4.2.2 CONFIGURAÇÃO

Esta fase é definida por comportamentos que ocorrem uma única vez no ciclo de vida do *bot*, antes da primeira atualização do *malware*. Onde se obtém o arquivo de configuração.

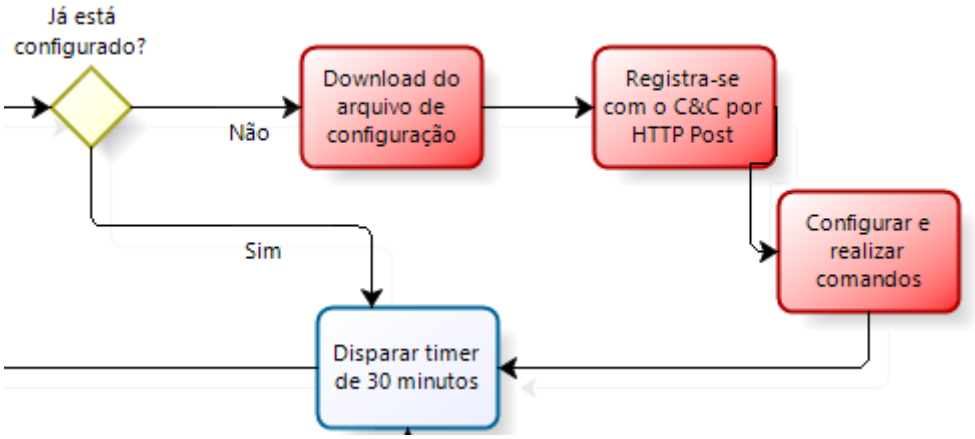


FIG 4.4. Diagrama de Transição – Configuração.

O arquivo que possui as configurações da *botnet* tem um conjunto de informações necessárias para o funcionamento do *bot* como, por exemplo, a lista de servidores (IP ou DNS), algoritmo de formação de nomes de domínios (DGA), temporização das requisições do *bot*, entre outros comandos de configurações internas da máquina infectada e roubo de dados (PACHECO, 2014).

Logo após a fase de Recém-Infecção o arquivo de configuração é requerido pelo novo *peer* para um dos *peers hard-coded*, se o arquivo for encontrado, então ocorre o *download* do arquivo e uma mensagem de sucesso é enviada. Somente a partir desta fase, o novo *bot* pode começar a receber instruções e realizar atualizações corretamente.

4.2.3 ATUALIZAÇÃO E KEEP-ALIVE

A atividade do *bot* nessa fase consiste de um laço que se repete a cada 30 minutos, sendo responsável por manter a lista de *peers* e de *proxies* atualizadas. Em cada iteração, é feito um pedido para cada um dos vizinhos pelas versões de seus

arquivos em binário e de configuração, servindo para checar se o mesmo está atualizado e se sua vizinhança está respondendo.

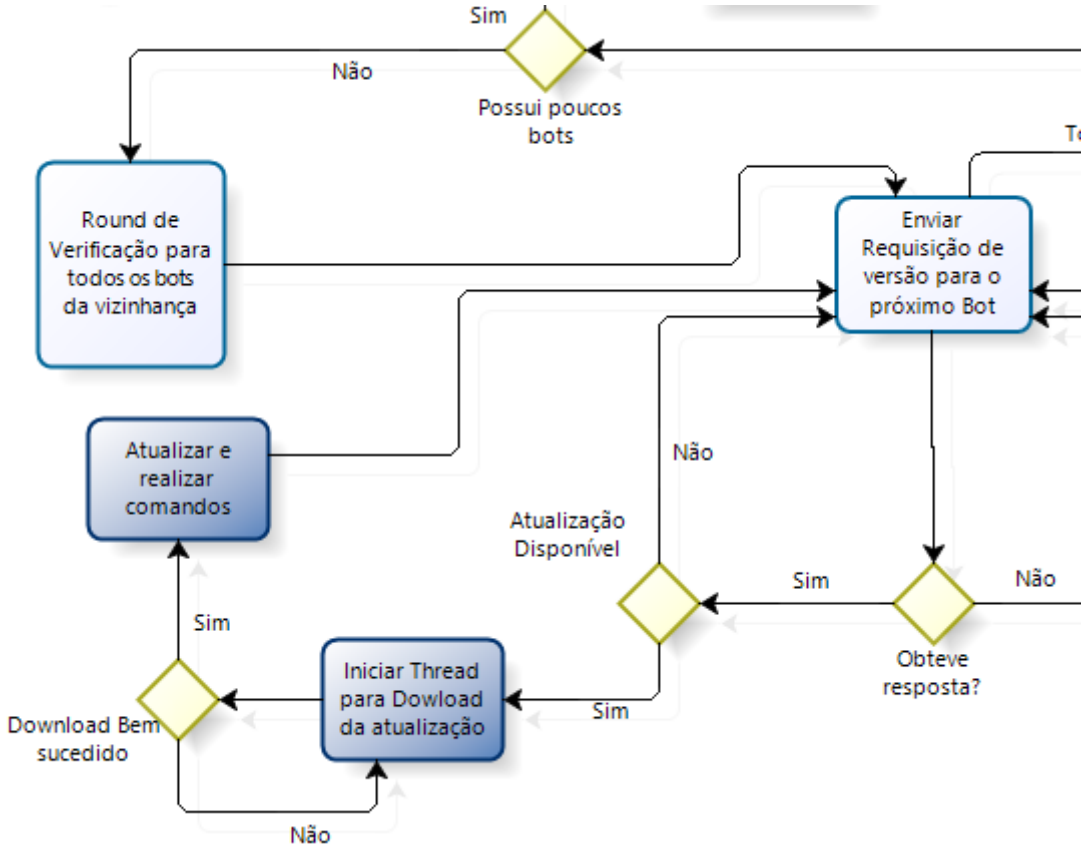


FIG 4.5. Diagrama de Transição – Atualização.

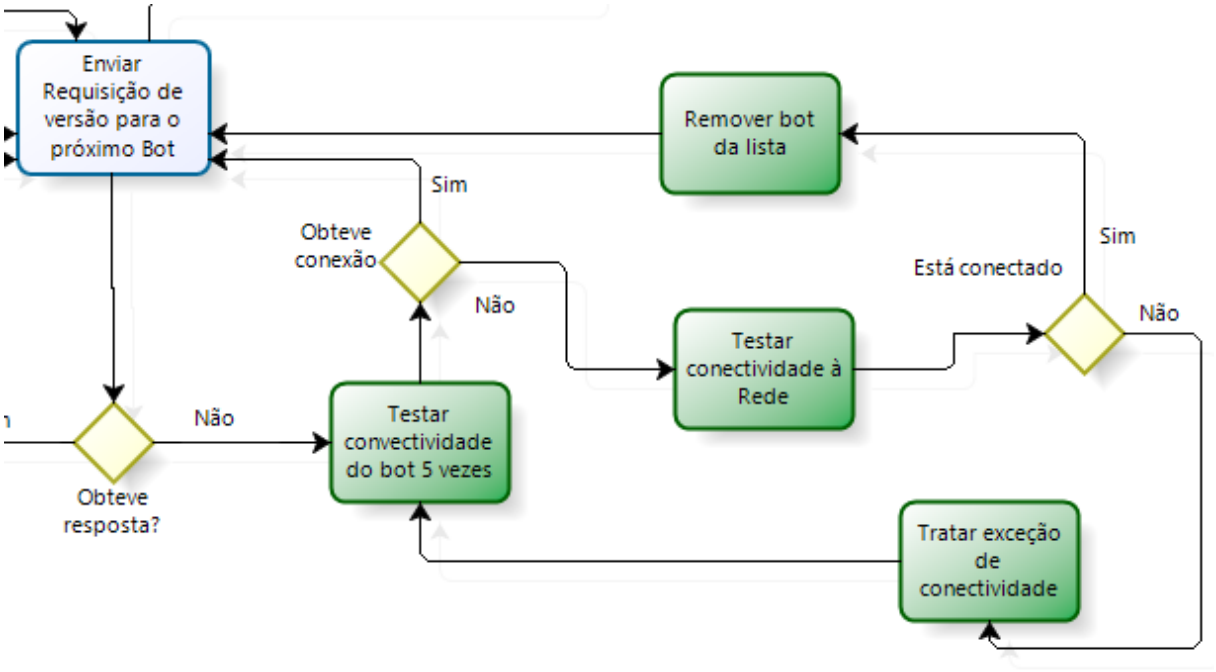


FIG 4.6. Diagrama de Transição – Keep-Alive

Quando o *bot* responde e possui uma versão mais recente, é feito o *download*

desta versão em um novo thread pelo *bot* receptor.

Após requerer uma versão mais recente da lista de *peers*, cada *peer* tem 5 chances para responder *request*, se ele não responder, então há uma suspeita que este *bot* não está mais em execução. Antes de descartá-lo, a Zeus checa sua conexão com a Internet tentando acessar o site “google.com” ou “bing.com” para descobrir se o problema do *request* não é uma falha de conexão de sua parte, se comprovar que sua conexão com a Internet está funcionando o *bot* é descartado da lista.

5 DESENVOLVIMENTO

Neste capítulo do trabalho, será tratado o desenvolvimento do projeto, ou seja, será descrito e explicado como foi feita a geração de traces a partir da modelagem feita. Será dividido em 2 etapas:

- Especificação;
- Implementação;

Em cada tópico será detalhado o que será abordado em cada uma das etapas.

5.1 ESPECIFICAÇÃO

Neste tópico do trabalho, será tratada a definição do problema, ou seja, será detalhado o que será feito na simulação da rede Zeus P2P. Esta definição será obtida a partir de duas etapas principais:

- Levantamento e análise de requisitos; e
- Especificação de requisitos.

A primeira etapa tem como objetivo gerar a Descrição geral do sistema e a segunda, o Modelo de casos de uso.

5.1.1 LEVANTAMENTO E ANÁLISE DE REQUISITOS

O levantamento de requisitos foi realizado a partir de (ANDRIESSE, 2013) e (SILVA, 2012) onde obteve o comportamento da *botnet*, assim como as mensagens que esta utiliza. Durante este processo, foram identificados e detalhados os requisitos implementados por esta ferramenta.

Durante a análise de requisitos, foi feito o refinamento e a priorização dos requisitos obtidos a partir da análise do que melhor caracteriza o *trace* desta rede no ambiente real.

Desta forma, a fase de levantamento e análise de requisitos consolidou o documento Descrição geral do sistema, presente no Apêndice 1 deste trabalho.

5.1.2 ESPECIFICAÇÃO DE REQUISITOS

A partir da Descrição geral do sistema, foi consolidado o documento Modelo de casos de uso. Este documento é composto pelo Diagrama de casos de uso e pela Descrição do modelo de casos de uso (presente no Apêndice 2 deste trabalho).

Na **Figura 5.1(a)** está representado o Diagrama dos casos de uso e na **Figura 5.1(b)** está ilustrado a sequência de ações que o sistema toma para gerar o tráfego da rede.

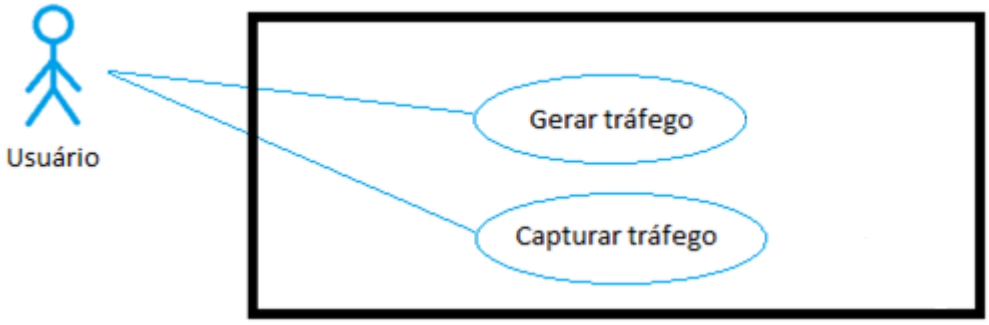


FIG 5.1(a) Diagrama de casos de uso

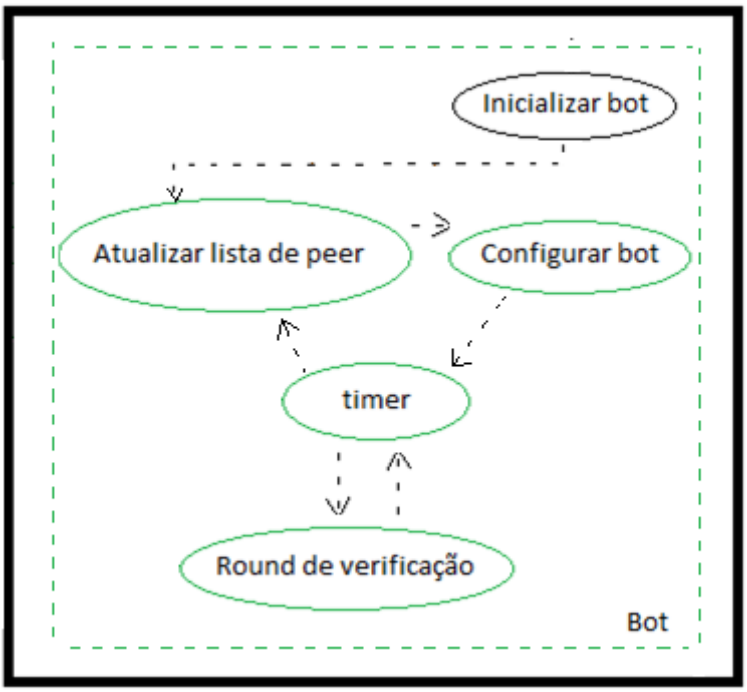


FIG. 5.1(b) Sequência das ações

Sendo assim, a fase de especificação definiu o que será feito para o desenvolvimento da *botnet* Zeus P2P a partir da consolidação do Modelo de casos de uso. Este modelo será a base para a próxima fase do trabalho, onde é definido como foi feito o desenvolvimento desta simulação.

5.2 IMPLEMENTAÇÃO

Neste tópico do trabalho será tratada a solução do problema, ou seja, será detalhado como foi feito o desenvolvimento da *Botnet* Zeus P2P, a partir do Modelo de casos de uso obtido na fase de Especificação. Este detalhamento foi obtido a partir das seguintes etapas:

- Modelagem das classes;
- Modelagem dos arquivos; e
- Uso do *Wireshark* para obtenção do trace simulado.

A primeira etapa tem como objetivo descrever as classes necessárias para o funcionamento do software. Já a segunda descreve os arquivos necessários para o bom funcionamento da *botnet*. E a terceira tem como objetivo descrever o uso da ferramenta *Wireshark* para a obtenção do tráfego simulado.

5.2.1 MODELAGEM DAS CLASSES

O simulador é composto de 8 classes responsáveis pelo seu funcionamento, *Botnet, LogWritter, Peer, TCPClient, TCPServer, UDPClient, UDPServer e XMLHandler* que serão descritas a seguir

- *Botnet*: classe onde são definidas as fases da *botnet* assim como todo o seu funcionamento;
- *LogWritter*: classe responsável por escrever o *log*, para assim ter-se um como comparar possíveis problemas de execução, ou até mesmo para comparar as ações tomados pelo *bot* com o tráfego gerado pela rede;
- *Peer*: classe auxiliar para armazenar a informações de *bots* vizinhos como suas portas, IPs e conexões, servindo de intermediário para tratar e se comunicar com *bots* vizinhos;

- TCPClient: classe responsável por enviar pedidos e interpretar respostas de mensagens TCP para *bots* vizinhos;
- TCPServer: classe instanciada no momento da inicialização do *bot*, responsável por interpretar mensagens TCP provindas de outros *bots*;
- UDPClient: classe responsável por enviar pedidos e interpretar respostas de mensagens UDP para *bots* vizinhos;
- UDPServer: classe instanciada no momento da inicialização do *bot*, responsável por interpretar mensagens UDP provindas de outros *bots*; e
- XMLHandler: classe responsável por ler e obter informações dos arquivos salvos no disco rígido.

Adicionalmente podem ser criados quantos *bots* forem necessários para a simulação da *botnet*, tanto internamente (localhost) como externamente, bastando explicitar o caminho dos arquivos que compõe o *bot*, que serão descritos a seguir, numa classe executável.

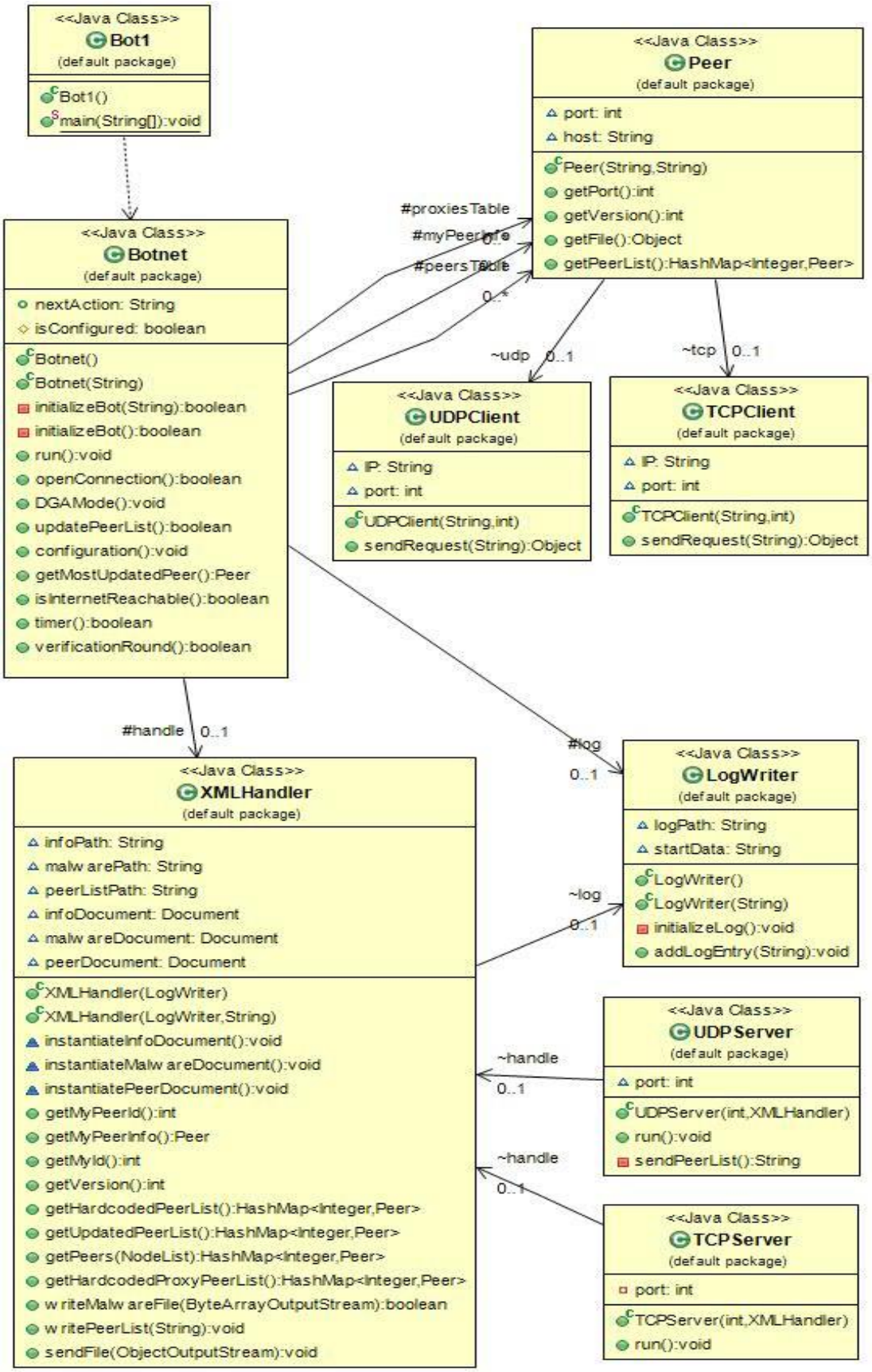


FIG.5.2. Diagrama de classes

Para iniciar a simulação, primeiro precisa-se instanciar o *bot*, nesta instanciação temos duas opções:

- Botnet() - construtor padrão da classe *Botnet* aonde o caminho dos arquivos que o *Bot* precisará é definido como “C:\BotSimulator\”; ou
- *Botnet* (String) - construtor alternativo aonde pode especificar o caminho dos arquivos.

Posteriormente basta executar o método público da classe *Botnetrun():void* para rodar indeterminadamente ou *run(int):void* para um certo número de minutos. Nestes métodos o simulador entrará num loop aonde a cada passo do laço verifica qual é a próxima ação a se seguir, ação esta que é modificada pelo último método chamado. Em futuras implementações, pode-se adicionar novos métodos modificando fluxo normal das ações, lembrando de no final do novo método configurar a variável global *nextAction* para a próxima ação desejada.

Uma observação a respeito da implementação é que se mais de um *bot* for instanciado internamente será necessário o uso de diferentes portas para cada um, assim como o uso de diferentes diretórios. No CD que está em anexo neste trabalho temos um exemplo de simulação com classes *Bot1*, *Bot2* e *Bot3* implementando estes métodos para melhor entendimento.

5.2.2 MODELAGEM DOS ARQUIVOS

Outro fator importante na modelagem do problema é em relação aos arquivos que são transmitidos e ou persistem nos *bots, sendo estes* similares aos de um ambiente real da *botnet* Zeus P2P. Na presente simulação, é necessário cobrir tanto a etapa de inicialização do *bot*, como as de *keep-alive* e de configuração, foi necessário um arquivo para cada etapa, cujas as funcionalidades serão descritas a seguir.

- Info.xml: arquivo usado na inicialização do *bot*, contendo informações de IP, de portas e dos *peers hard-coded*;

- Config.xml: contém a versão do *malware* instalada no *bot*, é utilizado quando um *bot* deseja se atualizar com o *bot* vizinho, sendo transmitido por TCP, para o *bot* menos atualizado que por sua ver irá se atualizar; e
- PeerList.xml: arquivo populado com os *bots* vizinhos ativos de um *bot* com seus respectivos IPs e portas, sendo utilizado na troca de *lista de peers*da fase de *keep-alive* e atualizado como descrito na modelagem desta simulação.

Uma vez que estes arquivos estão configurados corretamente, já é possível simular o comportamento de um *bot* por sua simples instanciação.

6 SIMULAÇÃO

Para obter o trace da simulação foi utilizada a ferramenta *Wireshark*, aonde existe a possibilidade de configurar entre quais portas, IPs e protocolos a comunicação será gravada, par desta forma obter um trace específico entre quaisquer máquinas que se vinculem a um *bot* da *botnet*. Por fim, também se utilizou esta ferramenta para salvar este trace para futuras consultas, ou até mesmo comparar com os arquivos de *log* gerado pela nossa aplicação confirmando que o *trace* obtido está de acordo com o especificado nas bibliografias pesquisadas, obtendo desta forma um modo de criar material de consulta para mais pesquisas nessa área.

No caso de implementar todos os *bots* na mesma máquina, é preciso utilizar outra ferramenta, a WinpCap, para capturar o tráfego do *loopback*, contudo ela possui a limitação de não captar o tráfego TCP para comunicações internas.

6.1 ANÁLISE DE RESULTADOS

Buscando não somente simular, mas também testar e validar todos os ramos do diagrama de fases implementado, foi desenvolvido uma simulação que contempla todo o sistema, sendo esta executada passo a passo, de forma a investigar tanto o trace gerado como o comportamento do sistema em diversas situações.

Para iniciar a simulação, foram introduzidos 4 *bots* ao sistema com suas características descritas a seguir.

- **Bot 1:**
    - Status de conexão à internet: *Online;*
    - Porta UDP aberta: 5051;
    - Versão do arquivo de configuração: 1;
    - Lista de *bots* vizinhos: *Bot* 2, *Bot* 3 e *Bot* 4.
- **Bot 2:**
    - Status de conexão à internet: *Online;*
    - Porta UDP aberta: 5052;

- Versão do arquivo de configuração: 2;
- Lista de *bots* vizinhos: *Bot* 1.

- **Bot 3:**
  - Status de conexão à internet: *Online;*
  - Porta UDP aberta: 5053;
  - Versão do arquivo de configuração: 3;
  - Lista de *bots* vizinhos: *Bot* 1.
- **Bot 4:**
  - Status de conexão à internet: *Offline;*
  - Porta UDP aberta: 5054;
  - Versão do arquivo de configuração: 4;
  - Lista de *bots* vizinhos: *Bot* 1.

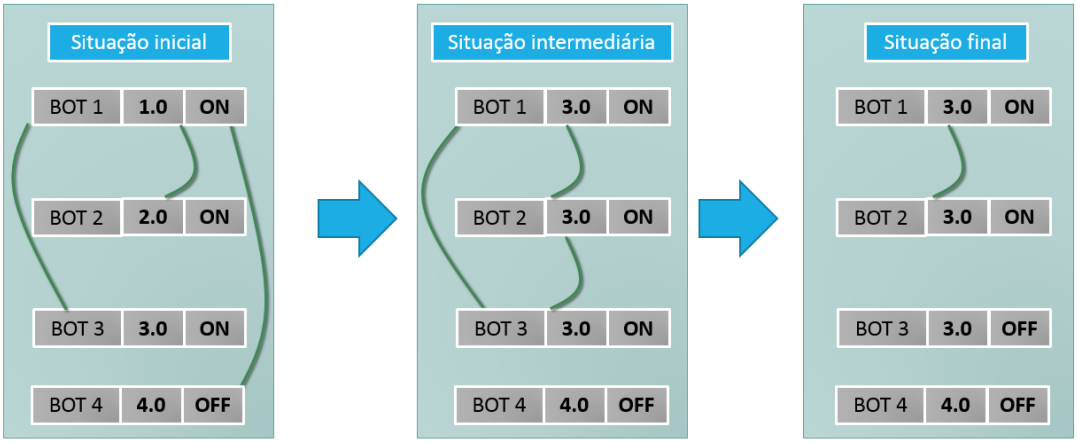


FIG.6.1. Diagrama de estados da simulação

A **Figura 6.1** representa 3 estados que a simulação irá passar. O estado intermediário é alcançado após o sistema se estabilizar, nesse estado os *bots* ativos passaram a se conhecer estando também atualizados. A partir desse momento, a execução do *Bot* 3 é interrompida com o intuito de testar a remoção de *bots* não responsivos, chegando assim no estágio final, onde apenas os *Bots* 1 e 2 estavam ativos.

A partir da definição dos estados de simulação, foi possível realizar a simulação da *botnet*, podendo ser verificado as ações de um *bot* em todas as fases, anteriormente modeladas.

No início da simulação os *Bots* 1, 2 e 3 abriram suas conexões UDP e TCP adicionando localmente os *bots hard-coded* para futuramente testar sua conectividade, já o *Bot* 4 por não estar conectado à internet terminou sua execução. Observa-se que ao decorrer desta etapa nenhum trace foi gerado, sendo o log do *Bot* 1 desta fase apresentado a seguir.

20_54_43: My Peer info1, Port:5051, Host:127.0.0.1
20_54_43: Opening connections
20_54_43: Getting hard coded peers
20_54_43: Peer 2 (5052, 127.0.0.1) founded
20_54_43: Peer 3 (5053, 127.0.0.1) founded
20_54_43: Peer 4 (5054, 127.0.0.1) founded
20_54_43: Proxy Table is empty
20_54_43: Bot initialization fully completed

FIG.6.2. Log da inicialização do *bot*

Em seguida cada *bot* entrou na fase de "Atualizar a lista de *Peers*" aonde enviaram uma mensagem UDP *version request* para cada um de seus *bots* vizinhos, identificando assim quais estavam ativos, e assim pedir sua lista de *peers com a mensagem UDP peer list request*.

O log desta fase do *Bot* 1 é apresentado na **Figura 6.3**, aonde observa-se que apenas os *bots* 2 e 3 estão responsivos e que o *Bot* 1 os pede suas peerlist para aumentar sua rede de nós.

20_54_43: Running action: updatePeerList
20_54_43: Hard coded peer 2 is responsive, add it to peers table and get peerList!
20_54_43: Hard coded peer 3 is responsive, add it to peers table and get peerList!
20_54_53: Peer 4 is not responsive, remove it!
20_54_53: Getting peerlist (127.0.0.1, 5052)
20_54_53: Getting peerlist (127.0.0.1, 5053)
20_54_53: Saving peer list to file

FIG.6.3. Log da fase de Atualização da lista de *peers*

Na **Figura 6.4**, é possível identificar a comunicação do Bot 1, com os Bots 2, 3 e 4, identificando que mensagens foram enviadas ao observa-se os campos '*Src Port'*, '*Dst Port'* e '*Text'*.

```
   139 127.0.0.1  127.0.0.1  UDP 38 Source port: 51175  Destination port: ita-manager
⊞ User Datagram Protocol, Src Port: 51175 (51175), Dst Port: ita-manager (5052)
⊟ Data (10 bytes)
    Data: 676574566572736 96f6e
    Text: getVersion
    [Length: 10]
   140 127.0.0.1  127.0.0.1  UDP 29 Source port: ita-manager  Destination port: 51175
⊞ User Datagram Protocol, Src Port: ita-manager (5052), Dst Port: 51175 (51175)
⊟ Data (1 byte)
    Data: 33
    Text: 3
    [Length: 1]
   141 127.0.0.1  127.0.0.1  UDP 38 Source port: 51176  Destination port: rlm-disc
⊞ User Datagram Protocol, Src Port: 51176 (51176), Dst Port: rlm-disc (5053)
⊟ Data (10 bytes)
    Data: 676574566572736 96f6e
    Text: getVersion
    [Length: 10]
   142 127.0.0.1  127.0.0.1  UDP 29 Source port: rlm-disc  Destination port: 51176
⊟ Data (1 byte)
    Data: 33
    Text: 3
    [Length: 1]
   143 127.0.0.1  127.0.0.1  UDP 38 Source port: 51177  Destination port: 5054
⊞ User Datagram Protocol, Src Port: 51177 (51177), Dst Port: 5054 (5054)
⊟ Data (10 bytes)
    Data: 676574566572736 96f6e
    Text: getVersion
    [Length: 10]
   144 127.0.0.1  127.0.0.1  ICMP      66 Destination unreachable (Port unreachable)
⊟ Internet Control Message Protocol
    Type: 3 (Destination unreachable)
    Code: 3 (Port unreachable)
    Checksum: 0x2d66 [correct]
  ⊞ Internet Protocol Version 4, Src: 127.0.0.1 (127.0.0.1), Dst: 127.0.0.1 (127.0.0.1)
  ⊞ User Datagram Protocol, Src Port: 51177 (51177), Dst Port: 5054 (5054)
  ⊟ Data (10 bytes)
      Data: 676574566572736 96f6e
      Text: getVersion
      [Length: 10]

280 127.0.0.1  127.0.0.1  UDP 43 Source port: 63513  Destination port: ita-manager
⊞ User Datagram Protocol, Src Port: 63513 (63513), Dst Port: ita-manager (5052)
⊟ Data (15 bytes)
    Data: 706565724c69737452657175657374
    Text: peerListRequest
281 127.0.0.1  127.0.0.1  UDP 45 Source port: ita-manager  Destination port: 63513
⊞ User Datagram Protocol, Src Port: ita-manager (5052), Dst Port: 63513 (63513)
⊟ Data (17 bytes)
    Data: 3120353035312031323 72e302e302e310a
    Text: 1 5051 127.0.0.1\n
282 127.0.0.1  127.0.0.1  UDP 43 Source port: 63514  Destination port: rlm-disc
⊞ Internet Protocol Version 4, Src: 127.0.0.1 (127.0.0.1), Dst: 127.0.0.1 (127.0.0.1)
⊞ User Datagram Protocol, Src Port: 63514 (63514), Dst Port: rlm-disc (5053)
⊟ Data (15 bytes)
    Data: 706565724c69737452657175657374
    Text: peerListRequest
283 127.0.0.1  127.0.0.1  UDP 45 Source port: rlm-disc  Destination port: 63514
⊞ User Datagram Protocol, Src Port: rlm-disc (5053), Dst Port: 63514 (63514)
⊟ Data (17 bytes)
    Data: 3120353035312031323 72e302e302e310a
    Text: 1 5051 127.0.0.1\n
```

FIG 6.4. *Trace* da fase de Atualização da lista de *peers*

Na implementação da *botnet*, a mensagem *version request* (UDP) é definida com o valor ‘*getVersion’* no campo ‘*Text’* e a mensagem peer list request (UDP) com o valor *‘peerListRequest’.* Sendo a descrição do trace desta etapa descrito a seguir.

- **Linha 139:** *Version request* (UDP) do *Bot* 1 para o *Bot* 2.
    - Porta de origem: 51175 (Valor arbitrário utilizado pelo *Bot* 1).
    - Porta é de destino: 5052 (Porta UDP do *Bot* 2).
    - Campo de texto: *getVersion.*
- **Linha 140**: *Version reply* (UDP) do *Bot* 2 para o *Bot* 1,
    - Porta de origem: 5052.
    - Porta é de destino: 51176 (*Bot* 1).
    - Campo de texto: *3 (versão do* Bot 2).
- **Linha 141:** *Version request* (UDP) do *Bot* 1 para o *Bot* 3.
    - Porta de origem: 51176 (Valor arbitrário utilizado pelo *Bot* 1).
    - Porta é de destino: 5053 (*Bot* 3).
    - Campo de texto: *getVersion.*
- **Linha 142**: *Version reply* (UDP) do *Bot* 3 para o *Bot* 1,
    - Porta de origem: 5053.
    - Porta é de destino: 51177 (*Bot* 1).
    - Campo de texto: *3 (versão do* Bot 3).
- **Linha 143:** *Version request* (UDP) do *Bot* 1 para o *Bot* 4.
    - Porta de origem: 51177 (Valor arbitrário utilizado pelo *Bot* 1).
    - Porta é de destino: 5054.
    - Campo de texto: *getVersion.*
- **Linha 144**: Mensagem ICMP, pois o *Bot* 4 não está responsivo.
- **Linha 280**: *Peer list request* (UDP) do *Bot* 1 para o *Bot* 2,
    - Porta de origem: 63513 (Valor arbitrário utilizado pelo *Bot* 1).
    - Porta é de destino: 5052.
    - Campo de texto: *peerListRequest.*

- **Linha 281:** Peer list *reply* (UDP) do *Bot* 2 para o *Bot* 1.
    - Porta de origem: 5052.
    - Porta é de destino: 63513
    - Campo de texto: 1 5051 127.0.0.1 (ID, porta e IP de um *peer* do *Bot* 2)
- **Linha 282**: *Peer list request* (UDP) do *Bot* 1 para o *Bot* 3,
    - Porta de origem: 63514 (Valor arbitrário utilizado pelo *Bot* 1).
    - Porta é de destino: 5053.
    - Campo de texto: *peerListRequest.*
- **Linha 283:** Peer list *reply* (UDP) do *Bot* 3 para o *Bot* 1.
    - Porta de origem: 5053.
    - Porta é de destino: 63514.
    - Campo de texto: 1 5051 127.0.0.1 (ID, porta e IP de um *peer* do *Bot* 3)

Em seguida dá-se início a fase de configuração aonde o *Bot* 1 envia *request* para os *bots* vizinhos 2 e 3 para saber qual está mais atualizado e assim mandar um *request* TCP pelos arquivos de configuração mais modernos, ações estas descritas no log desta fase na **Figura 6.5**.

20_54_53: Running action: configuration
20_54_53: Checking connectivity with Peer 2
20_54_53: Peer 2 connected with version 3
20_54_53: Checking connectivity with Peer 3
20_54_53: Peer 3 connected with version 3
20_54_53: Post to Controller Command
20_54_53: Configuring bot

FIG.6.5. Log da fase de Configuração

Nesta fase, contudo, não foi possível apresentar no *trace* da **Figura 6.6** a troca de pacotes por TCP, pois a simulação foi realizada em uma única máquina utilizando um *loopback,* e como mencionado anteriormente o método de captura WinpCap não consegue captar este tipo de pacote, sendo esta comprovação feita pela inspeção da versão do bot após o envio do *request* TCP. Vale ressaltar que em um ambiente real com várias máquinas o próprio *Wireshark* poderia captar os pacotes TCP.

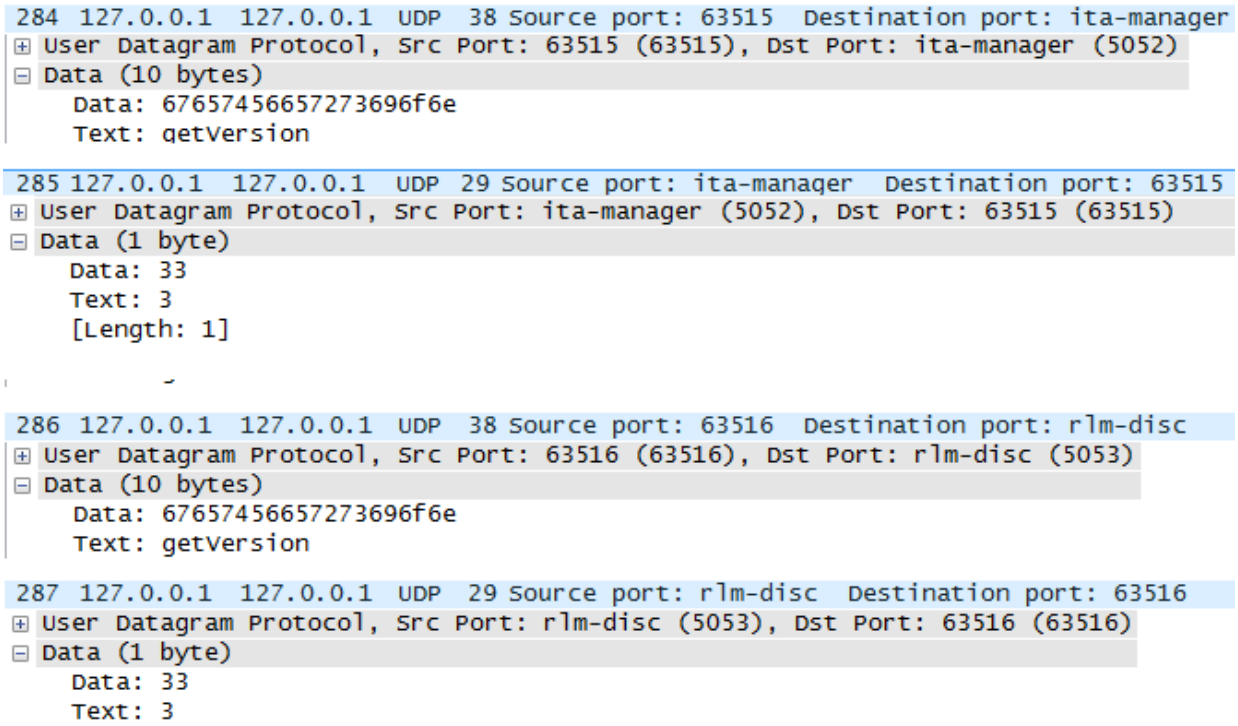


FIG. 6.6. *Trace* da fase de Configuração

As linhas 284 a 287 deste logo são similares as da 139 a 142, e o resultado é mesmo, o Bot 1 após enviar um *version request* para os *bots* 2 e 3, linhas 284 e 286 respectivamente, recebe os respectivos *version reply* destes *bots,* linhas 285 e 287.

Na próxima fase, a de atualização e *keep-alive*, também foi verificado o correto funcionamento do sistema pelo *trace* gerado, não sendo este apresentado aqui pela similaridade com o trace da **Figura 6.6**. Este trace se dá pelo pedido das versões dos arquivos e posteriormente pela transferência do arquivo por TCP, alcançando desta forma o estado intermediário do sistema e mantendo-se no laço de verificação enquanto os 3 *bots* estiverem ativos.

20_55_53: Running action: verificationRound
20_55_53: Checking connectivity with Peer 2
20_55_53: Peer 2 connected with version 3
20_55_53: Checking connectivity with Peer 3
20_55_53: Peer 3 connected with version 3
20_55_53: Bot up to date

FIG.6.7. Log do *round* de Verificação

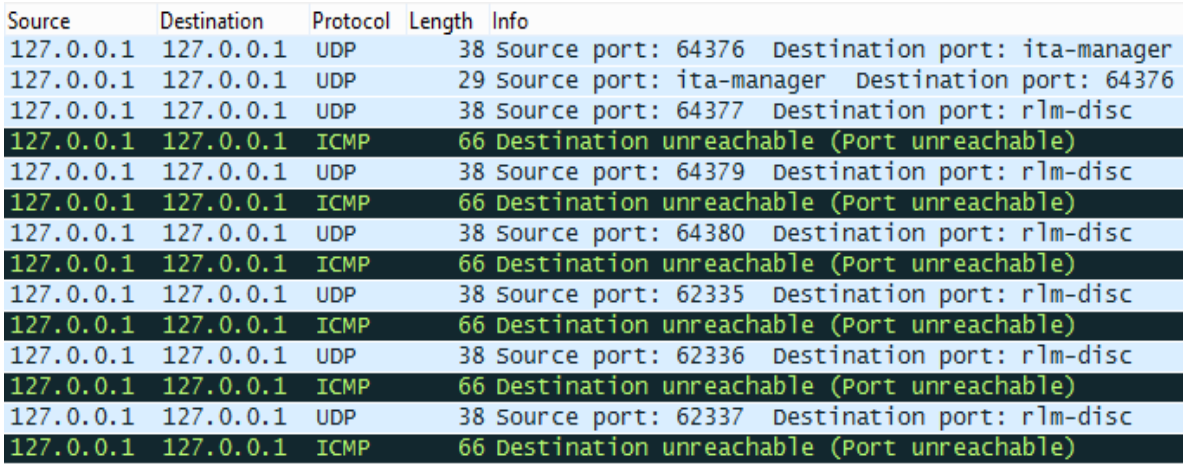
No log apresentado na **Figura 6.7**, os dois bots vizinhos estavam responsivos e

o Bot 1 estava atualizado, não acrescentando muito à verificação e validação do sistema. Desta forma, a fim de testar um novo cenário, foi desconectado um dos bots vizinhos para assim testar o comportamento do sistema quando há bots vizinhos não responsivos na fase de atualização.

Após parar a execução do *Bot* 3, obteve-se o *trace* da **Figura 6.9**, aonde o *Bot* 1, após o primeiro *version request* falhar, tenta se conectar mais 5 vezes com o *Bot* 3, o removendo de sua lista de *peers* ativos. Observe na **Figura 6.8** que após executar o timer o sistema recai para atualização de sua lista de *peers* devido à insuficiência de *bots* vizinhos ativos.

20_56_28 : Running action: verificationRound
20_56_28 : Checking connectivity with Peer 2
20_56_28 : Peer 2 connected with version 3
20_56_28 : Checking connectivity with Peer 3
20_56_38 : Peer 3could not be reached
20_56_38 : 1 try to connect
20_56_48 : 2 try to connect
20_56_58 : 3 try to connect
20_57_08 : 4 try to connect
20_57_18 : 5 try to connect
20_57_29 : Check if this bot is Online
20_57_29 : bot is online, remove unresponsive peer
20_57_29 : Bot up to date
20_57_29 :
20_57_29 : Running action: timer
21_00_29 :
21_00_29 : Running action: updatePeerList
21_00_29 : Peer 2 is responsive, add it to peers table and get peerList!
21_00_39 : Peer 3 is not responsive, remove it!
21_00_49 : Peer 4 is not responsive, remove it!
21_00_49 : Getting peerlist (127.0.0.1, 5052)
21_00_49 : Saving peer list to file

FIG. 6.8. Log da remoção de *peers* não-responsivos



Source	Destination	Protocol	Length	Info
127.0.0.1	127.0.0.1	UDP	38	Source port: 64376 Destination port: ita-manager
127.0.0.1	127.0.0.1	UDP	29	Source port: ita-manager Destination port: 64376
127.0.0.1	127.0.0.1	UDP	38	Source port: 64377 Destination port: rlm-disc
127.0.0.1	127.0.0.1	ICMP	66	Destination unreachable (Port unreachable)
127.0.0.1	127.0.0.1	UDP	38	Source port: 64379 Destination port: rlm-disc
127.0.0.1	127.0.0.1	ICMP	66	Destination unreachable (Port unreachable)
127.0.0.1	127.0.0.1	UDP	38	Source port: 64380 Destination port: rlm-disc
127.0.0.1	127.0.0.1	ICMP	66	Destination unreachable (Port unreachable)
127.0.0.1	127.0.0.1	UDP	38	Source port: 62335 Destination port: rlm-disc
127.0.0.1	127.0.0.1	ICMP	66	Destination unreachable (Port unreachable)
127.0.0.1	127.0.0.1	UDP	38	Source port: 62336 Destination port: rlm-disc
127.0.0.1	127.0.0.1	ICMP	66	Destination unreachable (Port unreachable)
127.0.0.1	127.0.0.1	UDP	38	Source port: 62337 Destination port: rlm-disc
127.0.0.1	127.0.0.1	ICMP	66	Destination unreachable (Port unreachable)

FIG. 6.9. As 6 tentativas de conexão ao *bot* 3

Similar a falha de tentativa de conexão com o *Bot* 4 na fase de atualização de lista de *peers*, na **Figura 6.9** também podemos verificar que a porta não pode ser alcançada como era esperado, sendo por isso o *Bot* 3 removido da lista de *peers* do *Bot* 1.

Com o fim desta simulação conclui-se que a implementação do sistema descrita neste capítulo foi bem sucedida aonde de fato conseguiu-se com sucesso chegar ao estado esperado descrito na **Figura 6.1** no início deste capítulo e, desta forma, gerar o *trace* da *botnet* Zeus P2P.

7 CONCLUSÃO

O presente trabalho teve por objetivo criar um ambiente para simular o funcionamento de uma rede de computadores infectada por *malwares*, para desta forma conseguir gerar de forma fidedigna *traces* de uma arquitetura Zeus P2P, a qual foi tomada como base.

A primeira etapa do trabalho – Entender o que é uma *botnet* e sua realidade atual, tomando como referência a *botnet* Zeus P2P. Além de entender seu contexto no mundo acadêmico. Ambos os tópicos são vistos nos capítulos 2 e 3, nos quais introduz-se as *botnets* e se entende a Zeus P2P, respectivamente.

A segunda etapa do trabalho – Modelaram-se as fases de vida dos *bots* na rede. Essa etapa é vista no capitulo 4, onde toma-se como foto as fases de Recém-Infecção, Configuração, Atualização e Keep-Alive.

A terceira etapa do trabalho –Também no capitulo 4 modela-se a comunicação entre os *bots*, necessária para atender os requisitos do trabalho.

A quarta etapa do trabalho – Nessa etapa elabora-se a infraestrutura necessária para atender os requisitos para que a rede exista. Sendo esses requisitos encontrados no capitulo 5 e a infraestrutura vista no capitulo 6 e 7.

A quinta etapa do trabalho – Responsável pela geração de código do trabalho, para que, de maneira otimizada, construísse a rede e a comunicação entre os *bots* propriamente dita. Além disso, obtem-se os resultados: os *traces* do tráfego gerado da comunicação entre os *bots*.

Com isso, a sociedade acadêmica é mais bem abastecida, dando mais condições para estudar essas redes maliciosas, as quais começam a crescer e difundir problemas para usuários e companhias. Como sugestão de trabalhos futuros, indica-se a expansão do sistema, abordando outras arquiteturas, para que a detecção e defesa contra ataques maliciosos sejam realizados de forma mais eficaz.

8 REFERÊNCIAS BIBLIOGRÁFICAS

ARANTES, A.C. Detecção e remoção de *botnets* na rede do POP-MG. Relatório Final de Projeto Orientado – Universidade Federal de Minas Gerais, 2006.

ANDRIESSE, D et al. *An Analysis of the Zeus Peer-to-Peer Protocol, 2013.* URL http://www.few.vu.nl/~da.andriesse/papers/zeus-tech-report-2013.pdf. Último acesso em: Março de 2014.

BACHER, P. et al. *Honeynet Project and Research Alliance. Know your enemy: Tracking*, 2008. URL http://www.honeynet.org/papers/bots/. Último acesso em: Outubro de 2013.

BREWSTER, T. *The evolution of the botnet*, 2010. URL http://www.itpro.co.uk/627487/the-evolution-of-the-botnet. Último acesso em: Outubro de 2013.

CHOI, H. et al. BotGad*: Detecting botnets by capturing group activities in network traffic. Proceedings of the Fourth International ICST Conference on COMmunication System software and middleware*, COMSWARE, 2009. pp.2:1-2:8.

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

EDUARDO, V.R. e COSTA, R.D. Modelagem e Simulação do tráfego *bot.* Iniciação à Pesquisa – Instituto Militar de Engenharia, 2013.

FEILY, M. etal. A survey of Botnet and Botnet Detection. *Third International Conference on Emerging Security Information, Systems and Technologies,* 2009.n.48, p.268-273.

HUDAK, T. An introduction into the World of Botnets, 2006. URL http://www.hudakville.com/infosec/. Último acesso em: Outubro de 2013.

NETO, R.P.C. Sistema de Detecção de Intrusos em ataques oriundos de *Botnets* utilizando método de detecção híbrido. Dissertação (Mestrado em Engenharia de Eletricidade) – Universidade Federal do Maranhão, 2011.

PACHECO, C. 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, 2014.

PERLIN, T. et al. Detecção de anomalias em redes de computadores através de transformadas Wavelet. *Revista Brasileira de Computação Aplicada,* 2011. n.1, p. 02-15.

PURI, R. *Bots&Botnet: An Overview. SANS Institute InfoSec Reading Room,* 2003.

SABAHI, F. e MOVAGHAR, A. Intrusion Detection: A survey. *The Third International Conference on Systems and Network Communications,* 2008. pp. 23-26.

SAHA, B. e GAIROLA, A. *Botnet*: An Overview. *CERT-In White Paper, Indian Computer Emergency Response Team,* 2005.

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), 2013. p. 1:1–6,

SILVA, S.S.C. etal.Botnets: A Survey.*Computer Networks*.n.57, 2012. pp. 378-403.

STANIFORD,S. etal.*How to own the internet in your spare time. Proceedings of the 11th USENIX Security Symposium,* 2002.p. 149 and 167.

STONE-GROSS, B. *The Lifecycle of Peer-to-Peer (Gameover) Zeus*, 2012. URL http://www.secureworks.com/cyber-threat-intelligence/threats/The_Lifecycle_of_ Peer _ to_Peer_Gameover_ZeuS/. Último acesso em: Março de 2014.

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*, 2008. p. 1–24. Springer. ISBN 978-0-387-68768-1. URL http://dblp.uni-trier.de/db/series/ais/ais36.html.

TYAGI, A. K. e AGHILA, G. *A wide scale survey on botnet. International Journal of Computer Applications* (0975 - 8887),2011. Volume 34 - Number 9.

ZHU, Z. et al. Botnet Research Survey. *Computer Software and Applications.32nd Annual IEEE International.pp.967-972, 2008.*

ZEINDALOO, H.R. etal. A taxonomy of botnet detection techniques.*3rd IEEE International Conference on Computer Science and Information Technology* (ICCSIT), vol. 2, 2010. pp. 158–162.

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.

9 APÊNDICES

9.1 APÊNDICE 1 – DESCRIÇÃO GERAL DO SISTEMA

1. Gerar Tráfego
    - O usuário poderá gerar tráfego de uma rede Zeus P2P maliciosa. O sistema precisa de uma rede de computadores ou máquinas virtuais e suas respectivas portas e IPs. O Usuário configura os *bots* que irão gerar o tráfego e preenche os arquivos de configuração. O sistema inicia a simulação.
    - *Bots* mal configurados: se os *bots* estiverem mal configurados o erro é escrito no log e o programa termina.
2. Capturar Tráfego
    - O usuário poderá salvar o trafego gerado por meio do *Wireshark*. O sistema precisa de um filtro. O usuário fornece o filtro e inicia a captura do tráfego. O sistema salva o tráfego em arquivo.
3. Inicializar *bot*
    - No início da simulação o sistema lê os arquivos de configuração dos *bots*, abre as conexões TCP e UDP, e inicializa o arquivo de LOG. O Sistema persiste as informações dos arquivos no sistema
    - *Bots* mal configurados: se os *bots* estiverem mal configurados o erro é escrito no log e o programa termina.
4. Atualizar Lista de *Peers*
    - Um *bot* poderá atualizar sua lista de *peers* quando necessário. O *bot* checa sua conectividade com os *bots* vizinhos e com os *hard-coded*. Os *bots* vizinhos que não responderem serão removidos. Os *bots* que responderem enviaram sua lista de *bots* vizinhos. O *bot* receptor adiciona os novos *bots* que estiverem ativos.
    - *Bot* não possui o número mínimo de *bots* vizinhos ativos: entra em modo DGA até que consiga o número mínimo de *bots* vizinhos ativos.
5. Configurar *Bot*

- O *bot* envia uma mensagem para todos os *bots* vizinhos pedindo a versão do *malware*. Os *bots* vizinhos respondem a versão do *malware*. O *bot* receptor envia uma requisição para o *bot* vizinho mais atualizado. O *bot* mais atualizado envia os arquivos de configurações para o *bot* receptor. O *bot* receptor atualiza sua versão.
- *Bot* vizinho não responde: O *bot* que está requerendo a versão remove o *bot* não responsivo de sua lista de *bots* ativos e continua.

6. Timer
    - O *bot* espera 30 minutos, verifica se tem o número mínimo de *bots* vizinhos para seu funcionamento e entra no *round* de verificação.
    - *Bot* não tem o número mínimo de *bots* ativos após a verificação: entra no modo de atualização de lista de *peers*.
7. *Round* de Verificação
    - O *bot* verifica a conectividade de todos os *bots* vizinhos enviando uma requisição pela versão do *malware*. Se ele responder o armazena-se a versão do *bot*, para no fim do *round* de verificação fazer o *download* da versão mais atualizada do *bot* caso encontre uma versão mais atualizada do *malware*.
    - Se um *bot* não responder ao pedido ele tenta se conectar mais 5 vezes, o removendo da lista de *bots* vizinhos ativos caso não seja possível ele não responder.

9.2 APÊNDICE 2 – DESCRIÇÃO DO MODELO DE CASOS DE USO

CdU 1

- **Nome: Gerar Tráfego**
- Ator primário: Usuário
- Ator secundário: Não há
- Pré-condições: Arquivos de configuração devem estar presentes
- Fluxo básico de eventos:
    1. Este caso de uso começa quando o usuário deseja iniciar a simulação

2. O usuário escolhe por quanto tempo a simulação vai rodar e executa o programa.
3. O sistema termina a simulação, fecha as portas de comunicação e o caso de uso termina.

- Fluxo alternativo de eventos:
  - *BOTS MAL CONFIGURADOS*: No passo 2 do FBE, se os *bots* estiverem mal configurados
    1. Sistema apresenta a mensagem Erro e termina sua execução.
  4. Pós-condições: Sistema produz o tráfego da *botnet* e arquivos de log.
- Outras informações:
  - Arquivos de configuração: *Config.xml e Info.xml*
  - *Portas de comunicação: UDP e TCP*

CdU 2

- **Nome: Capturar Tráfego**
- Ator primário: Usuário
- Ator secundário: Não há
- Pré-condições: Não há
- Fluxo básico de eventos:
  1. Este caso de uso começa quando o usuário deseja capturar o tráfego
  2. Usuário inicia o sistema do *Wireshark* e passa os parâmetros de busca.
  3. Sistema começa a monitorar o tráfego da rede
  4. Usuário executa o caso de uso “Gerar Tráfego” e quando este acaba para o monitoramento.
  5. Sistema salva o tráfego em um arquivo e o caso de uso termina
- Fluxo alternativo de eventos: Não há
- Pós-condições: Sistema possui o tráfego capturado
- Outras informações:
  - Parâmetros de busca: portas e IP dos *bots*.