ISSN:1983 7402 ITA, 29 SET a 01 OUT de 2015

MADEC: Arquitetura multiagente imunoinspirada para identificar atividades de exfiltração de dados ocultas por *rootkits*

Rafael Salema Marques, Paulo André Lima de Castro
Instituto Tecnológico de Aeronáutica (ITA) – Praça Marechal Eduardo Gomes, 50, Vila das Acácias, São José dos Campos – SP – Brasil.

***Resumo* — Este trabalho apresenta uma arquitetura imunoinspirada baseada na Teoria do Perigo e no cruzamento de informações de tráfego de rede percebido entre agentes em uma rede local. A presente pesquisa busca identificar atividades de exfiltração de dados executada por códigos maliciosos furtivos que são capazes de ocultar seu tráfego de rede do computador hospedeiro. A abordagem demonstrou eficácia ao identificar tráfegos ilícitos desconhecidos pelo Sistema Operacional subvertido. Além disso, a arquitetura não demanda conhecimento prévio sobre características ou comportamentos do código malicioso e tampouco necessita avaliar o conteúdo das mensagens trafegadas, capacidade desejável em um cenário em que o código malicioso é desconhecido pelos mecanismos de defesa e faz uso de criptografia para proteger suas comunicações.**

***Palavras-Chave* — *Malware*, Sistema Imune Artificial, *rootkit*, Sistema Multiagente.**

I. INTRODUÇÃO

Botnets [1] são ameaças cibernéticas globais utilizadas como ferramentas para viabilizar atividades maliciosas tendo como alvo pessoas e organizações. Conforme estatística divulgada pelo *Federal Bureal of Investigation* (FBI), as *botnets* causaram um prejuízo na economia global de mais de 110 bilhões de dólares em 2014 e infectam 18 computadores por segundo no mundo [2].

Algumas *botnets* mais elaboradas possuem características furtivas que buscam acobertar atividades maliciosas na máquina comprometida, essa capacidade é conhecida como *rootkit* [3][4]. *Rootkits* são um conjunto de técnicas e funcionalidades que permitem ocultar e assegurar a presença de um invasor ou de outro código malicioso em um computador comprometido [5]. Dentre as diversas capacidades furtivas que um *rootkit* pode proporcionar a um código malicioso, podemos citar a ocultação do tráfego de rede do computador hospedeiro.

Existem dois tipos de *rootkit* quanto a privilégios de acesso, [6] *user mode* e *kernel mode*. Em um *rootkit user mode*, o código de execução não tem acesso diretamente a memória ou a qualquer recurso de *hardware*. Porém em um *rootkit kernel mode*, o código terá acesso total e irrestrito ao hardware. Esse último será capaz de enviar qualquer instrução à CPU bem como fazer referência a endereços de memória física. O acesso ao *kernel mode* é reservado para o nível mais baixo, onde a maioria das funções de confiança do sistema operacional são executadas.

Os desenvolvedores de *malware* [3] estão criando técnicas inovadoras relacionadas a furtividade cada vez mais sofisticadas [7][8][9]. Essas inovações possibilitam a subversão do sistema operacional de forma tão profunda e inusitada que sua detecção pelas soluções disponíveis no mercado torna-se uma tarefa de improvável sucesso, e esse cenário é agravado quando o código malicioso faz parte de uma campanha *Advanced Persistent Threat* (APT) [10], onde o atacante é suportado por uma organização, possui conhecimento técnico elevado, e é capaz de desenvolver um código malicioso furtivo direcionado especificamente para ludibriar as defesas do alvo.

Podemos definir como um dos piores cenários quando o código malicioso adquire privilégios *kernel mode* e desta forma passa a não ser detectável pelas soluções de segurança disponíveis. Nesse caso, o invasor seria capaz de controlar e utilizar remotamente todos os recursos disponíveis para o Sistema Operacional. Além de uma campanha APT, esse panorama crítico é extremamente plausível de ocorrer em uma máquina infectada por uma *botnet* implementada com características de *rootkit*.

O objetivo deste trabalho é propor uma arquitetura multiagente [11][12] capaz de identificar atividades maliciosas de exfiltração de dados, nos casos em que o *malware* passa despercebido pelas defesas locais, possui características furtivas de *rootkit* e busca ocultar seu tráfego em um sistema comprometido. Reconhecemos que é fundamental fazer uso de soluções que busquem proteger o sistema de infecções, porém a experiência diz que não há proteção perfeita. Para tanto, utilizaremos uma abordagem imunoinspirada que cruza informações de tráfego de rede local a fim de identificar padrões que indiquem suporte a atividades maliciosas no momento em que tentam exfiltrar dados ou procuram fechar canais de Comando e Controle (C2) [1][13].

Esse SMA faz uso de metáforas imunológicas, enquadrando no conceito de um Sistema Imune Artificial (SIA).

SIAs são compostos por metodologias inteligentes, inspiradas no sistema imunológico biológico, para a solução de problemas do mundo real [3]. São pautados em funcionalidades e mecanismos de defesa presentes no sistema imune humano, tais como caráter distribuído, adaptabilidade, memória, reconhecimento de padrões, resiliência dentre outras características desejáveis na modelagem de sistemas de engenharia.

ISSN:1983 7402

II. TRABALHOS RELACIONADOS

Em sua grande maioria, as abordagens para detecção de atividades maliciosas executadas pelas *botnets* são orientadas pelo tipo de canal de C2 utilizado [1], fazendo uso de análise de anomalia de tráfego de rede [14], com especial atenção aos protocolos HTTP, IRC [15][16] e DNS [17][18].

Entretanto, levando em consideração o pior caso citado na introdução, essas abordagens se tornam ineficazes, pois o *malware* será capaz de ocultar seu tráfego da máquina local.

Em [19], é proposta uma abordagem Multiagente que considera a implementação de *rootkits*, onde busca cruzar diversas informações na máquina. Entretanto essa abordagem é dependente de conhecimento prévio das técnicas utilizadas pelos atacantes e limitada a um Sistema Operacional específico.

Foram citados diversos trabalhos em [1] que abordam a detecção de *botnets* baseada em comportamento cooperativo de múltiplos sensores, porém desconsideram a capacidade de ocultação de informações de tráfego de rede por parte dos códigos maliciosos.

III. FUNDAMENTAÇÃO TEÓRICA

A. TEORIA DO PERIGO

Em nosso trabalho utilizou-se a Teoria do Perigo (*Danger Theory* – DT) [20][21], onde a resposta imune é desencadeada por estímulos que o corpo reconhece como danoso. Essa teoria ainda afirma que as células ao sofrerem danos ou morte de forma não natural, emitem sinais no tecido que podem ser identificados por células apresentadoras de antígeno (*Antigen Presenting Cells* – APC).

As Células Dendríticas (DC) são um tipo de APC que estão espalhadas pelos tecidos, e são capazes de perceber diversos sinais e informações moleculares. Além disso, são responsáveis por interpretar os dados coletados e encaminhar essa informação às células do sistema imune responsáveis pelo início da resposta imunológica no corpo humano, o Linfócito T. Essa interação celular inspirou o Algoritmo das Céluas Dendríticas, (*Dendritic Cell Algorithm* – DCA).

B. ALGORITMO DAS CÉLULAS DENDRÍTICAS

DCA é um algoritmo desenvolvido para detecção de anomalias. Foi criado por [22] onde foi feita uma descrição minuciosa do algoritmo. Sua primeira implementação como protótipo foi apresentada em 2005 [23] e posteriormente de forma completa em um sistema capaz de detectar atividades maliciosas em tempo real no ano de 2006 [24].

O DCA apresenta vantagens ao ser aplicado em sistemas que buscam detectar falhas de segurança em tempo real, pois requer pouco poder computacional e um curto período de treinamento. Para isso, faz uso de Células Dendríticas Artificiais (aDC) que serão responsáveis por processar a informação de sinais e evidências fazendo referência ao comportamento da DC biológica.

A aDC processará os dados recebidos e poderá se tornar madura caso perceba sinais de perigo que irão subsidiar a decisão de uma resposta imune. Por outro lado, também poderão se tornar semimaduras, situação que suprime a defesa imunológica.

Seguindo a mesma proposta, porém com foco na redução do número de parâmetros de entrada e a viabilidade de execução com menos recursos computacionais, foi vislumbrada uma versão determinística do DCA, nominada dDCA (*Deterministic Dendritic Cell Algorithm* – DCA) [25].

C. ALGORITMO LINFONODO

Uma parte importante da nossa defesa imunológica acontece no Sistema Linfático [26]. É um sistema constituído por vasos similares às veias (vasos linfáticos) que acompanham paralelamente o aparelho cardiovascular e coletam o líquido que envolve as células nos tecidos.

Além de outras funcionalidades, esse Sistema é responsável por eliminar microrganismos ou outras partículas que possam causar dano ao corpo humano por meio de uma filtragem executada nos órgãos linfáticos, dentre eles os Linfonodos. Os Linfonodos são estruturas pequenas, ovaladas, encapsuladas e interpostas no trajeto dos vasos linfáticos onde ocorre a filtragem da linfa. Eles estão distribuídos pelo corpo humano e apresentam uma grande concentração de Linfócitos T no seu interior. Basicamente as DCs que foram expostas a sinais no tecido são conduzidas ao linfonodo pela linfa, apresentam os sinais percebidos aos Linfócitos T que por sua vez irão desencadear ou não uma resposta imune.

O local em que ocorre a filtragem e a interação celular inspiraram o algoritmo batizado como Linfonodo proposto no presente trabalho, pois faz referência à interação celular e a filtragem que ocorrem no interior deste órgão linfático, onde uma entidade do sistema é responsável por coletar o tráfego (DC), encaminhar os dados percebidos para outra entidade (Linfócito T), que por sua vez analisará as informações e buscará identificar padrões que indiquem atividades maliciosas em curso, (perigo) para enfim tomar as providências cabíveis (resposta imunológica). Podemos ainda considerar o algoritmo Linfonodo uma variante do dDCA, porém com mudanças na metáfora e no tratamento dos dados.

Quanto à metáfora no dDCA, a análise dos sinais é executada durante o processo de amadurecimento das DCs, enquanto que em nossa pesquisa, o foco se dá nas ações ocorridas dentro do linfonodo, onde as DCs que trafegam pela linfa – ao adentrar o linfonodo – apresentam seus sinais aos Linfócitos T que irão processar as informações percebidas.

Em relação ao algoritmo, o tratamento dos sinais foi modificado conforme equações que serão apresentadas a seguir, onde foi despendida uma especial atenção aos sinais similares percebidos pelo sistema, que para o problema em questão são de vital importância no processo decisório das entidades envolvidas.

Tendo em vista que buscou-se desenvolver um algoritmo de detecção de anomalias em tempo real que solucionasse o problema de exfiltração de dados por *malwares* não detectados pela máquina local, e fosse aplicável a qualquer rede que utilize o protocolo de comunicação IP (*Internet Protocol*) [27], em nosso protótipo os sinais serão representados por pacotes transmitidos pelas conexões de

rede e filtrados pelo sistema no momento em que abandonam os limites da rede local. Os pacotes contém cabeçalhos que guardam informações de data, hora, IP de origem, IP de destino e porta de destino.

No algoritmo Linfonodo dividiu-se os sinais filtrados em três tipos:

1) *Sinais Lícitos (SL)*: São sinais liberados no tecido quando uma célula sofre uma morte programada (apoptose), onde não ocorre a ruptura da membrana plasmática. É um sinal que indica ausência de perigo.
2) *Sinais Ilícitos (SI)*: São sinais liberados no tecido por uma morte não programada da célula, de forma que há ruptura da membrana plasmática (calor, radiação, frio, trauma, bactérias, etc.). A concentração desse sinal indica ao sistema imune a presença de um invasor causador de dano.
3) *Sinais Similares (SS)*: É um sinal que apresenta similaridade com outro sinal já conhecido pelo sistema (mesmos IP de origem, IP de destino e porta de destino).

Para que possamos mensurar o grau de anomalia detectado pelo algoritmo, é necessário definir um número limite de sinais por execução. Em nossa implementação foi definida a quantidade de 10.000 pacotes filtrados por execução, que metaforicamente corresponderá ao ciclo de vida do Linfócito T.

A seguir serão apresentadas as equações relativas ao tratamento de sinais:

$$Ts = \left(\sum_{SI} * W_{SI}\right) + \left(\sum_{SL} * W_{SL}\right) \quad (1)$$

$$I_a = \frac{\left(\sum_{SI} * W_{SI}\right) * \left(\max\left(\sum_{SSi}\right) * W_{SS}\right)}{T_s} \quad (2)$$

Onde em (1) SL e SI são os sinais de entrada e W_{SI} e W_{SL} são os pesos aplicados aos somatórios de SI e SL respectivamente. O valor obtido pelo cálculo de Ts (Total de sinais) será comparado ao limite de sinais por execução (ciclo de vida) utilizado como condição de parada.

Em (2) I_a é um sinal de saída que corresponde ao índice de anomalia observado. SS_i corresponde a sinais similares ilícitos e W_{SS} é o peso aplicado a SS_i que equivale à maior quantidade de sinais similares ilícitos. Na Tabela I apresentamos os pesos de sinais utilizados no protótipo da presente pesquisa que foram parametrizados por observações estatísticas durante os experimentos.

TABELA I PESOS DOS SINAIS PROPOSTOS

Pesos	Valores
W_{SL}	1
W_{SI}	8
W_{SS}	4

Aliado ao algoritmo Linfonodo, nesse trabalho utilizamos o princípio da detecção por extrusão [28]. Esse conceito é essencial para o funcionamento do modelo proposto, onde a técnica de defesa não pretende impedir o código malicioso de comprometer o sistema, mas identificar suas tentativas de exfiltrar informações e operar a máquina comprometida.

D. ARQUITETURA MULTIAGENTE PROPOSTA: MADEC

Um agente é uma entidade que percebe o seu ambiente e tem um comportamento autônomo em consequência de suas observações de forma a satisfazer seus objetivos [29]. Um Sistema Multiagente (MAS) é um conjunto de agentes que interagem em um ambiente comum. Em nossa pesquisa foi implementado um MAS, complexo e benevolente [30] batizado como MADEC (*Multi-Agent Data Exfiltration Catcher*). Benevolente pois há interações cooperativas na produção de conhecimento entre os agentes, e complexo porque foram idealizados 2 tipos específicos de agentes. Sendo assim foram vislumbradas cinco tarefas específicas para os agentes do sistema:

1) Coleta de dados: Mecanismo de aquisição das evidências que serão utilizadas na tarefa de análise dos comportamentos dos participantes da rede.

2) Armazenamento: Tarefa que viabiliza salvaguardar as informações coletadas pelos agentes, de forma que todos os participantes da rede que irão executar a tarefa de análise possam acessá-la posteriormente.

3) Análise: Tratamento dos dados armazenados referentes às entidades participantes da rede com a finalidade de identificar atividades maliciosas.

4) Alerta: Notificar os demais agentes do sistema em relação ao conhecimento produzido pela análise.

5) Isolamento: Essa tarefa se aplica a todas as entidades participantes do Sistema. Consiste em um processo decisório individual dos agentes embasado nos padrões identificados pela tarefa de análise de forma a identificar e bloquear as comunicações com um nó não íntegro.

As tarefas supracitadas foram distribuídas em 2 tipos de agentes que serão apresentados a seguir:

1) Agente Coletor (AC): Responsável por executar as tarefas de Coleta de dados, Armazenamento e Isolamento. Esse agente deve ser executado em todos os computadores da rede local. Metaforicamente faz parte do papel da DC ao coletar informações e sinais nos tecidos do corpo humano.

No contexto do MADEC, esse agente que será executado com privilégios *user mode*, e irá atualizar um banco de dados quando solicitado pelo Agente Auditor – entidade que será apresentado a seguir – com dados de conexões externas à rede local (data, hora, IP de destino e porta de destino) que é capaz de perceber no momento da solicitação.

2) Agente Auditor (AA): Esse agente faz uso do conceito de *Firewall* distribuído descrito por [31] que contraria a premissa que o lado interno da rede é confiável. Na solução adotada, o AA foi instalado em uma máquina localizada na saída da rede local onde está implementado um *Firewall* Transparente com *Bridge*.

Esse posicionamento estratégico do AA é necessário pois dessa forma o fluxo do tráfego da rede local pode ser auditado e controlado por esse agente. Sendo assim, o sistema é capaz de cruzar informações de tráfego externo que recebe da rede interna com o tráfego percebido pelos AC. Caso o AC não perceba uma conexão externa feita pela máquina hospedeira (SI), pode-se classificar essa situação como um forte indício de comprometimento por um código malicioso que oculta seu tráfego de rede do computador onde o AC está

instalado (Perigo). A Fig. 1 ilustra o fluxo de comunicação dos agentes que será elucidado a seguir:

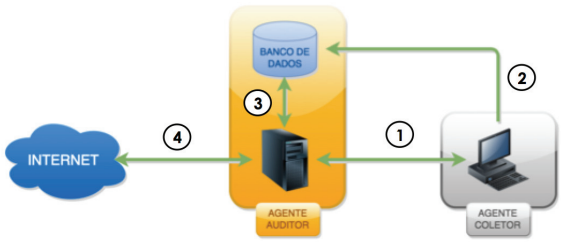

Fig. 1. Fluxo de comunicações entre AA e AC.

Ao perceber uma conexão externa proveniente da máquina que hospeda o AC, (marcação 1) o AA faz uma consulta ao banco de dados (marcação 3) em uma lista de conexões conhecidas pelos AC integrantes do sistema, (*Whitelist*) e caso a conexão esteja presente na *Whitelist*, é feita uma consulta a lista de conexões desconhecidas pelo AC e percebidas pelo AA (*Blacklist*).

Caso a conexão não esteja presente na *Whitelist*, o AA solicitará uma atualização da *Whitelist* (marcação 2) com as conexões conhecidas pelo AC correspondente ao computador que originou a conexão em foco.

Ao término da atualização reportado pelo AC, o AA executará uma nova consulta à *Whitelist* (marcação 3). Se o AA encontrar a conexão na *Whitelist*, isso indica que o AC conhece a conexão e não há indícios de ocultação de tráfego. Porém, se não for encontrada, sugere que o AC desconhece a conexão, e aponta fortes indícios de atividade maliciosa ocultada por um *rootkit* e a conexão será inserida na *Blacklist*.

Por fim, se o cálculo de Ts for maior ou igual a quantidade de pacotes estipulados para um ciclo de vida, todas as entradas presentes na *Whitelist* são excluídas a fim de garantir uma boa performance em redes com muito tráfego e que fazem uso de equipamentos com restrições de processamento. Em seguida será realizado o cálculo de I_a, e caso exista uma quantidade de pacotes filtrados pelo AA com o mesmo perfil desconhecido pelo AC que resulte em um valor maior que a quantidade limite parametrizado (*threshold*), o AA alertará os demais nós e isolará o nó que originou a conexão desconhecida, e os AC por sua vez, decidirão se devem ou não isolar o nó indicado pelo alerta.

Neste momento cabe a observação que redes de computadores podem apresentar diversos tipos de atrasos [27]. Logo, existe a possibilidade do sistema identificar um SI equivocadamente devido a um atraso no processamento ou recebimento da informação de que se trata de uma SL. Essa situação é indesejável pois poderá culminar em um falso positivo. Por esse motivo, o sistema foi implementado de forma a desconsiderar um SI a partir do momento que recebe a informação atrasada que o SI se trata de um SL, ou seja, o AA recebe tardiamente a informação que o AC conhece a conexão que havia sido identificada como SI erroneamente, sendo removida da *Blacklist* e inserida na *Whitelist*.

Abaixo apresenta-se o pseudocódigo do algoritmo linfonodo descrito anteriormente:

```
conn: conexão externa (ip_orig, ip_dest, data, porta)
threshold: valor máximo de Ia que dispara o alerta e o isolamento
Ts: limite de pacotes por execução

if (conn está na Whitelist) {
   if (conn está na Blacklist)
      Remove(conn.blacklist);
} else {
   Solicita ao AC que cadastre as conn conhecidas na Whitelist;
   if (conn não está na Whitelist)
      Insere(conn.blacklist);
}
if (Ts >= limite de pacotes) {
   limpa Whitelist;
   calcula Ia;
   If (Ia > threshold) {
      disparar alerta e isolamento;
   }
}
```

IV. Implementação e Experimentos

O MADEC é um sistema que se propõe a realizar análise de tráfego em tempo real. Logo, buscou-se uma linguagem que viabilizasse a escalabilidade da aplicação e a ausência de gargalos de processamento devido a grande quantidade de pacotes que seriam tratados ao mesmo tempo. Sendo assim, foi escolhido a plataforma Node.js [32].

Node.js faz uso de um modelo assíncrono não-bloqueante para tratamento de entrada e saída (I/O) orientado a eventos. Essa característica viabiliza que a leitura e escrita em arquivos ou banco de dados não irá impedir a execução de outras tarefas na aplicação, fato que viabiliza um modelo leve e eficiente no tratamento de dados para aplicações de tempo real que fazem uso de recursos distribuídos pela rede.

A fim de suportar a tarefa de Armazenamento executada pelo AC, é necessário um banco de dados capaz de tratar dezenas de milhares de registros de forma rápida e eficiente. Dessa forma, optou-se pelo MongoDB [33], um banco de dados não relacional que é uma mistura entre os repositórios escaláveis baseados em chave/valor que traz consigo a tradicional riqueza de funcionalidades dos bancos relacionais.

Para que o AA seja capaz de capturar os pacotes que saem da rede local, foi utilizado o módulos pcap [34], que viabiliza a comunicação entre a biblioteca libpcap [35] e o Node.js.

Quanto à interações entre os agentes, foram feitas por serviços RPC (Remote Procedure Call) [36], que pode ser definido como um protocolo para execução remota de procedimentos em computadores ligados em rede.

A fim de validar o modelo proposto, o MADEC foi exposto a três tipos de cenário:

1) AC não comprometido: Nesse cenário o computador hospedeiro do AC não está comprometido por nenhum tipo *malware*, apresentando um comportamento normal do Sistema Operacional.

2) AC comprometido: Nesse cenário o computador hospedeiro do AC foi comprometido por um código malicioso com propriedades de rootkit de forma a não perceber uma conexão TCP externa à rede local para em uma porta remota específica.

3) AC comprometido inicialmente e não comprometido posteriormente: Esse cenário inicia sem qualquer atividade maliciosa e durante o vigésimo ciclo de vida, o computador hospedeiro será comprometido pelo mesmo *malware* do

segundo cenário, e a atividade maliciosa será encerrada durante o vigésimo quinto ciclo.

V. RESULTADOS

Nos três cenários descritos, o protótipo foi exposto a um tráfego médio de 300.000 pacotes (30 ciclos de vida) totalizando 900.000 pacotes analisados. Na Fig. 2, apresentam-se os valores de I_a ao término de cada ciclo de vida nos cenários propostos.

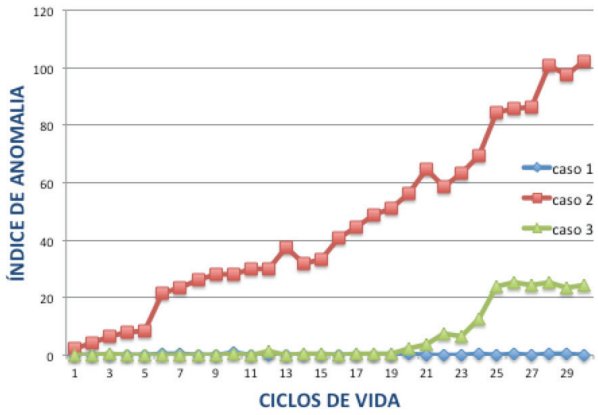

Fig. 2. Valores de I_a ao término de cada ciclos de vida.

Ao dar início a análise dos resultados, foi percebido que o valor máximo de I_a em um cenário livre de atividade maliciosa foi 1,322 e o valor mínimo observado em um cenário comprometido foi de 2,198. Sendo assim, baseado na amostragem obtida nos experimentos e buscando-se evitar falsos positivos, foi estipulado que o valor de I_a para que as tarefas de alerta e isolamento fossem executadas deveria ser igual a soma dos dígitos mais significativos dos dois números citados, obtendo-se o valor limite de I_a igual a 3 para que o sistema classifique o nó hospedeiro do AC como não confiável.

No primeiro caso, durante os ciclos, foi observado que por motivo de atrasos de processamento ou de entrega de mensagens, por vezes sinais lícitos foram identificados momentaneamente como ilícitos e posteriormente reclassificados corretamente. Fato que foi evidenciado ao final do vigésimo segundo ciclo, onde o sistema identificou 10.007 SL, 110 SI e 0 SS, gerando um I_a igual a 0,08. No ciclo seguinte foram identificados 9.886 SL, 4 SI e 2 SS produzindo um I_a igual a 0,026. Sendo assim, conclui-se que a capacidade de reclassificação implementada no algoritmo proporcionou resiliência a atrasos e confiabilidade na classificação, contribuindo para evitar falsos positivos.

No segundo cenário, o código malicioso enviou uma média de 3 pacotes por minuto, simulando um serviço de notificação de disponibilidade do nó para o atacante. Foi percebido que com essa pequena quantidade de tráfego, o sistema identificou a conexão ilícita durante o início do segundo ciclo. Essa detecção eficaz foi possível devido ao aumento significativo do valor de I_a quando o sistema percebe um incremento na concentração de SS.

Do quinto para o sexto ciclo do experimento foi simulada uma atualização do *malware*, que elevou I_a de 8,445 para 21,731. Logo, ficou evidenciado que a técnica percebe variações de tráfego que podem indicar certos tipos de atividades, como por exemplo atualização de código, download/upload de arquivos ou ainda um ataque DDoS [37].

No terceiro cenário, o sistema comportou-se de forma similar ao primeiro cenário até o vigésimo ciclo, momento em que as atividades maliciosas foram iniciadas. Após a infecção o gráfico apresentou similaridades com o segundo cenário, evidenciando consistência no modelo. A partir do vigésimo quinto ciclo foram encerradas as atividades maliciosas e o gráfico passou a apresentar um comportamento de estagnação, indicando que a atividade maliciosa ocorreu durante os ciclos de 20 a 25.

Devido ao fato da abordagem desconsiderar o conteúdo das mensagens trafegadas como evidências, essa proposta é aplicável a cenários onde o atacante faz uso de qualquer tipo de criptografia, diferentemente da maioria dos trabalhos citados na presente pesquisa.

VI. CONCLUSÃO

A pesquisa em pauta propõe uma arquitetura que utiliza o cruzamento de informações sobre as conexões que o computador comprometido percebe e o que realmente existe, a fim de identificar atividades maliciosas de ocultamento de tráfego em um computador quando esse último é incapaz de fazê-lo pelos recursos de defesa que dispõe.

O MADEC apresentou resiliência e confiabilidade quando submetidos a atrasos de redes e eficácia ao identificar tráfegos oriundos de atividades maliciosas nos casos específicos a que foi exposto. Além disso, o algoritmo Linfonodo apresentado não demanda conhecimento prévio sobre características ou comportamentos do código malicioso e tampouco conhecimento sobre o conteúdo das mensagens trafegadas para classificar as conexões, sendo capaz de identificar uma conexão ilícita a despeito da criptografia utilizada pelo atacante.

Entretanto, essa solução somente será eficaz nos casos em que o código malicioso busca ocultar seu tráfego da máquina local, técnica cada vez mais utilizada pelos desenvolvedores de *malware*.

Como proposta de trabalho futuro, sugere-se a realização de experimentos em outros cenários e algumas implementações nos ACs integrantes do MADEC, de modo a viabilizar uma verificação mútua em relação a padrões de anomalia típicos de levantamento de conhecimento sobre a disponibilidade de nós e serviços, bem como ataques de negação de serviço oriundos da rede local.

REFERÊNCIAS

[1] M. Bailey, E. Cooke, F. Jahanian, Y. Xu and M. Karir, "A Survey of Botnet Technology and Defenses", Proc. Cybersecurity Applications and Technology Conf. for Homeland Security, 2009.

[2] "Botnets infecting 18 systems per second, warns FBI", disponível em: <http://www.v3.co.uk/v3-uk/news/2355596/botnets-infecting-18-systems-per-second-warns-fbi> Acessado em 17 de junho de 2015.

[3] US-CERT "Understanding Hidden Threats: Rootkits and Botnets" disponível em: <https://www.us-cert.gov/ncas/tips/ST06-001> Acessado em 19 de junho de 2015.

[4] G. Hoglund , J. Butler, “Rootkits: Subverting the Windows Kernel”, Addison-Wesley, 2005.

[5] CENTRO DE ESTUDOS, RESPOSTA E TRATAMENTO DE INCIDENTES DE SEGURANÇA NO BRASIL. Cartilha de Segurança para Internet, 2ed., São Paulo: Comitê Gestor da Internet no Brasil, 2012. Disponível em: <http://cartilha.cert.br/livro/>. Acesso em: 18 junho 2015.

[6] Intel® 64 and IA-32 architectures software developer’s manual combined volumes: 1, 2A, 2B, 2C, 3A, 3B, and 3C, disponível em: <http://www.intel.com/content/dam/www/public/us/en/documents/manuals/64-ia-32-architectures-software-developer-manual-325462.pdf> Acessado em 21 de junho de 2015.

[7] DEFCON 21 “BoutiqueKit: Playing WarGames with Expensive Rootkits and Malware” disponível em: <https://www.youtube.com/watch?v=gKUleWyfut0> Acessado em 19 de junho de 2015.

[8] MalwareTech “Using Kernel Rootkits to Conceal Infected MBR” disponível em: <http://www.malwaretech.com/2015/01/using-kernel-rootkits-to-conceal.html> Acessado em 19 de junho de 2015.

[9] MalwareTech “MalwareTech SBK - A Bootkit Capable of Surviving Reformat” disponível em: <http://www.malwaretech.com/2015/06/hard-disk-firmware-rootkit-surviving.html> Acessado em 19 de junho de 2015.

[10] B. E. Binde, R. McRee and T. J. O’Connor, “Assessing Outbound Traffic to Uncover Advanced Persistent Threat”, SANS Technology Institute, 2011.

[11] M. Wooldridge, J. S. Rosenschein, “Autonomous Agents and Multi-Agent Systems”, Springer, 2002.

[12] R.H Bordini, M. Dastani, J. Dix, A. El Fallah Seghrouchni, “Multi-Agent Programming Languages, Platforms and Applications”, Springer 2005.

[13] G. Gu, J. Zhang, and W. Lee, “BotSniffer: Detecting Botnet Command and Control Channels in Network Traffic,” in Network and Distributed System Security, 2007.

[14] J. R. Binkley and S. Singh. An algorithm for anomaly based botnet detection. In Proceedings of USENIX Steps to Reducing Unwanted Traffic on the Internet Workshop (SRUTI), pages 43–48, July 2006.

[15] E. Cooke, F. Jahanian, and D. McPherson, “The zombie roundup: Understanding, detecting and disrupting botnets”. Usenix Workshop on Stepts to Reducing Unwanted Traffic on the Internet, Cambridge, MA, July 2005.

[16] A. Karasaridis, B. Rexroad, and D. Hoeflin, “Wide-Scale Botnet Detection and Characterization”, AT&T Labs, HotBots'07, 2007.

[17] The Honeynet Project. “Know your enemy: Tracking botnets.” disponível em: <http://www.honeynet.org/papers/bots/>, Acessado em 21 de junho de 2015.

[18] R. Villamarin-Salomon and J. Brustoloni, “Identifying Botnets Using Anomaly Detection Techniques Applied to DNS Traffic”, The Fifth IEEE Consumer Communications & Networking Conference (CCNC), January 2008.

[19] M. Szymczyk, “Detecting botnets in computer networks using multi-agent technology”. IEEE 4th Int. Conf. on Dependability of Computer Systems, p.192-201, 2009.

[20] P. Matzinger, “Tolerance, Danger, And The Extended Family”, Annu. Rev. Immunology, v. 12, p. 991-1045, 1994.

[21] U. Aickelin and S. Cayzer, “The Danger Theory and Its Application to Artificial Immune Systems,” In the Proceedings of 1st International Conference on Artificial Immune Systems (ICARIS), University of Kent at Canterbury, UK, Sept. 9–11, 2002.

[22] J. Greensmith, “Detecting Danger: The Dendritic Cell Algorithm”, PhD thesis, School of Computer Science, University Of Nottingham, 2007.

[23] J. Greensmith, U. Aickelin, and S. Cayzer. “Introducing Dendritic Cells as a novel immune- inspired algorithm for anomaly detection”, In Proc. of the 4th International Conference on Artificial Immune Systems (ICARIS), LNCS 3627, pages 153–167. Springer-Verlag, 2005.

[24] J. Greensmith, J. Twycross, and U. Aickelin, “Dendritic cells for anomaly detection”, .In Proc. of the Congress on Evolutionary Computation (CEC), pages 664–671, 2006.

[25] J. Greensmith and U. Aickelin, “The Deterministic Dendritic Cell Algorithm”, presented at 7th International Conference on Artificial Immune Systems, Phuket, Thailand, 2008.

[26] A. Stevens, J. Lowe, “Histologia humana”, 2.ed. São Paulo: Manole, 2001. p.117-135.

[27] S. A. Tanenbaum, “Redes de Computadores”, Editora Campus, 3ª edição, 1999.

[28] R. Bejtlich, “Extrusion Detection: Security Monitoring for Internal Intrusions”, Addison-Wesley Professional, 2005.

[29] M. Wooldridge, “An Introduction to MultiAgent Systems”, 2. ed. Glasgow: John Wiley and Sons, Ltd, Publication, 2009.

[30] M. Knapik, J. Johnson, “Developing Inteligent Agents for Distributed Systems”, NY, Computing McGraw-Hill, 1998.

[31] S. M. Bellovin, “Distributed Firewalls”, ;login: magazine, special issue on security, November 1999.

[32] Node.js, Disponível em: <https://nodejs.org/> Acessado em 09 de julho de 2015.

[33] MongoDB Disponível em: < https://www.mongodb.org/> Acessado em 09 de julho de 2015.

[34] Módulo pcap, “raw packet capture, decoding, and analysis”, Disponível em < https://www.npmjs.com/package/pcap> Acessado em 09 de julho de 2015.

[35] Libpcap, “The Libpcap Project”, Disponível em <http://sourceforge.net/projects/libpcap/> Acessado em 09 de julho de 2015.

[36] RFC 1057 RPC “Remote Procedure Call Specification” Version 2 Sun Microsystems, Inc., 1988.

[37] ICANN Report, DNS Distributed Denial of Service (DDoS) Attacks, Security and Stability Advisory Committee (SSAC), March 2006.