Aprimorando o simulador Simbo para testes de botnets avançadas

Fábio Harada Kubo, Moisés Danziger, Marco Aurélio Amaral Henriques

[1] Departamento de Engenharia de Computação e Automação Industrial (DCA)
Faculdade de Engenharia Elétrica e de Computação (FEEC)
Universidade Estadual de Campinas (Unicamp)
13.083-852 – Campinas, SP – Brasil

ra116740@fee.unicamp.br, {danziger, marco}@dca.fee.unicamp.br

Abstract. *This work aimed to improve the simulator Simbo that was designed for botnet simulation which make use of new attack techniques, highlighting those based on the use of machine learning. In order to reach that goal, we proposed a scenario that the bots act on infected hosts, filtering the network traffic and reporting relevant information to its command and control center with an intelligent engine that is responsible for making decisions. In the development of this scenario, it was noticed that its scalability would be compromised if there was no way to parallelize the work done by the bots. Thus, we developed a solution to distribute the simulation in several processing cores, increasing the simulator performance with a speedup of approximately 3.0. This solution will allow the study of new scenarios, among them we can cite the one in which the bots will also make use of computational intelligence techniques.*

Resumo. *Este trabalho teve como objetivo melhorar o simulador Simbo que foi projetado para simulações de botnets que façam uso de novas técnicas de ataque, destacando-se aquelas baseadas no uso de aprendizado de máquina. Para alcançar esse objetivo, propusemos um cenário no qual os bots atuam no host infectado filtrando o tráfego da rede e reportando informações relevantes para seu centro de comando e controle que, dotado de um motor inteligente, é responsável pela tomada de decisões. No desenvolvimento deste cenário, notou-se que sua escalabilidade seria comprometida se não houvesse uma forma de paralelizar o trabalho realizado pelos bots. Assim, desenvolvemos uma solução para distribuir a simulação em vários núcleos de processamento, obtendo-se um ganho de velocidade com fator de 3.0. Esta solução irá permitir o estudo de novos cenários, dentre eles, cita-se aquele em que os bots também farão uso de técnicas de inteligência computacional.*

1. Introdução

As botnets (conjunto de computadores infectados com um *software* que formam uma rede e que pode ser utilizada para a execução de ações maliciosas, como envio de *spams*, roubo de informações sigilosas, etc) são consideradas pragas virtuais que afetam milhões de usuários. A maioria deles não tem noção que seus dispositivos, com acesso a internet, estão infectados e fazem parte de uma botnet. Com o passar dos anos, os atacantes tem desenvolvido botnets mais robustas e resilientes. Na tentativa de combatê-las, o mercado

de segurança da informação aparenta sempre estar um passo atrás. Isso ocorre porque os atacantes buscam formas de inserir complexidade à detecção de suas botnet - também conhecido por técnicas de evasão. Dessa forma, o estudo de novos modelos de botnets é uma maneira de ajudar no desenvolvimento de novas ferramentas de detecção. Ou seja, se não é possível estar no mesmo passo dos atacantes, estimar quais serão seus próximos passos pode ajudar na busca por melhores ferramentas de segurança ou até mesmo atualizar as já existentes.

Nesse sentido, o conteúdo desse trabalho tem como base o aprimoramento de um simulador já existente de modo a permitir seu uso em simulações de botnets avançadas como aquela apresentada por Danziger e Henriques [Danziger and Henriques 2017] em que, através de um modelo teórico, os autores propuseram uma botnet que faz uso de técnicas de aprendizado de máquina (ML) como um motor inteligente e que a permite não ser dependente do botmaster para a tomada de decisões. Tal modelo implica em mudança de paradigma pois as botnets são conhecidas por sua dependência dos botmasters. Botnets que fazem uso de técnicas de ML na ponta (bots) ou no C2 (centro de comando e controle) não tem sido vistas ainda, embora não seja possível provar que não existam. Para que seja possível realizar experimentos com botnets sofisticadas é necessário um ambiente que seja totalmente controlado. Ambientes no qual o pesquisador possa realizar experimentos considerados perigosos já existem, porém, por envolver alto risco e complexidade, é importante que o usuário tenha conhecimento total do ambiente. Infelizmente, na maioria dos casos, as ferramentas existentes não são de código aberto e/ou não correspondem as necessidades dos pesquisadores. Por isso, nosso objetivo aqui é a implementação e/ou aprimoramento de novos módulos no simulador Simbo apresentado por Balabanian, Danziger e Henriques [Balabanian et al. 2016] para ser usado no contexto de novos modelos de botnets [1].

2. Trabalhos Relacionados

A simulação de botnets não é uma novidade, pode-se citar o trabalho de Agarwal [Agarwal 2010], onde foi realizado um estudo sobre a performance da botnet Storm, sendo utilizado o *framework* OMNeT++ [Varga 2001] para simular o comportamento da botnet a nível de pacotes. Há ainda os contínuos esforços de Kotenko, condensados em [Kotenko], que faz uso de sistemas multi-agentes para simular botnets, utilizando, também, o OMNeT++.

O presente trabalho diferencia-se dos outros dois pois pretende ser genérico o suficiente para que seja possível utilizá-lo no estudo de diversos tipos de botnets existentes e, até mesmo, modelos de botnets que estão, até onde se sabe, apenas na teoria.

3. Simbo e suas novas funcionalidades

O Simbo (código aberto) foi planejado para ser um simulador de novos modelos de botnets e outras pragas virtuais. Ele tem, como base principal, o simulador de eventos discretos OMNeT++ e o *framework* INET [INET 2017]. A Figura 1 apresenta a primeira versão da arquitetura do Simbo. Nela, é possível ver quatro módulos dentro do ambiente do OMNeT++. Apenas o módulo de armazenamento de dados estava fora do controle do OMNeT++. Após analisar a arquitetura inicial e confrontá-la com as limitações do OMNeT++

[1] Os desenvolvimentos apresentados neste trabalho se encontram no repositório: `https://github.com/regras/simbo_v2`

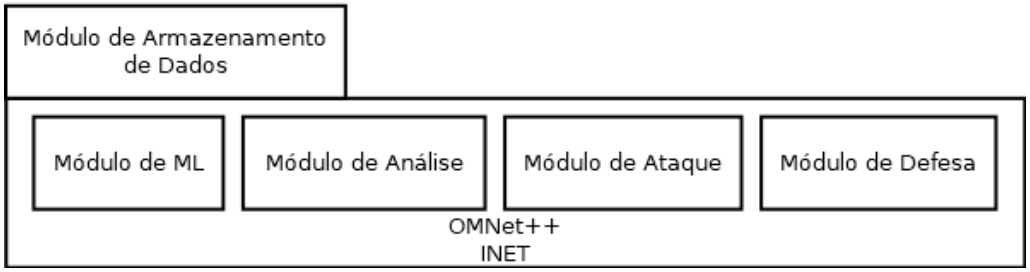

Figura 1. Arquitetura inicial do Simbo. Observe que os módulos estavam diretamente projetados para execução dentro do ambiente do OMNeT++.

decidimos modificá-la e criar um novo módulo, denominado módulo de comunicação, assim como retirarmos os módulos que estavam dentro do ambiente do OMNeT++ e os colocar acima do módulo de comunicação. Essa mudança permite maior flexibilidade no processo de desenvolvimento deixando o OMNeT++ apenas com o controle da infraestrutura da rede a ser simulada. A Figura 2 apresenta a nova arquitetura do Simbo. Os módulos mais escuros estão relacionados a atividades que foram desenvolvidas e/ou aprimoradas. Para os módulos em cinza claro foram desenvolvidas atividades parciais. Os módulos em branco não foram objeto de atividades no contexto deste trabalho.

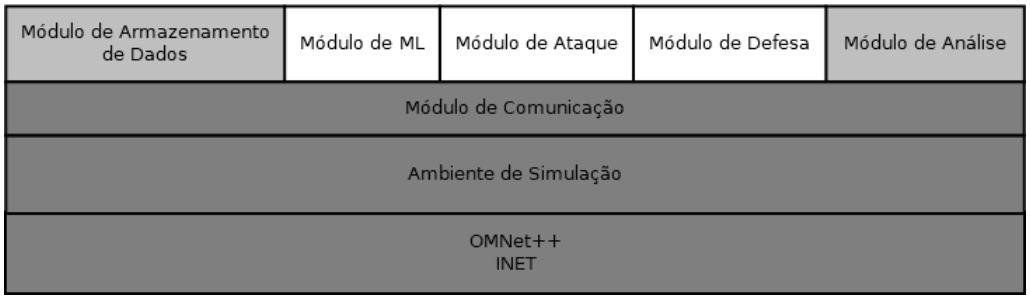

Figura 2. Arquitetura do Simbo, onde os módulos em retângulos pintados são os que foram beneficiados pelos aprimoramentos em curso.

3.1. Implementando o modelo de comunicação da botnet Athena

Antes de começar a visionar um cenário em que o Simbo pudesse simular o comportamento de uma botnet que faça uso de inteligência computacional, foi implementado no simulador o modelo de comunicação oriundo da botnet Athena [Athena] que é considerada uma botnet moderna. A base da implementação foi a versão que utiliza o protocolo HTTP e, dessa forma, a comunicação entre bot e C2 é feita do seguinte modo: (i) informações sobre o host infectado são coletados e colocados no formato *string*. Depois, a *string* gerada é codificada em base64 para, em seguida, com a utilização de duas *strings* geradas aleatoriamente, também codificadas em base64, realizar uma cifra de substituição, (ii) uma *string* de 24 bytes é gerada e utilizada como um marcador de dados pelo servidor ao produzir sua resposta. Esses valores passam por uma codificação URL (codificação por cento), para só então serem enviados ao C2 divididos em três parâmetros de uma requisição HTTP POST. O C2, ao receber tais dados, responde ao bot com uma lista de

comandos, definidos pelo botmaster, codificados em base64, prefixada com o marcador de dados enviado pela bot. As Figuras 3 e 4 resumem essa comunicação.

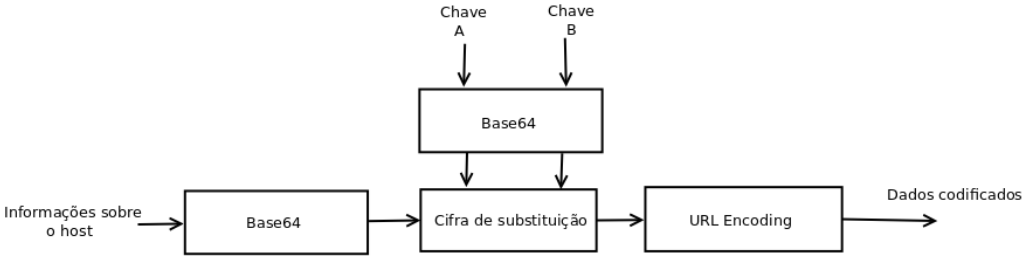

Figura 3. Codificação dos dados coletados.

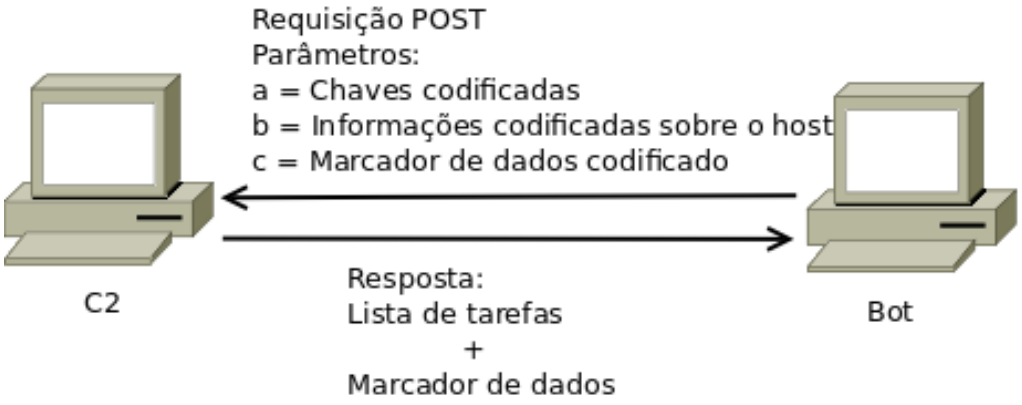

Figura 4. Fluxo de comunicação entre bot e C2.

3.2. Um novo modelo para as botnets

Buscando avaliar novos paradigmas, o Simbo deve ser capaz de simular o comportamento de uma botnet que contemple o uso de inteligência no C2 e no bot. Devendo ser capaz de simular tanto defesa quanto ataque, ou seja, existem três modos de operação: (i) ataque, (ii) defesa e, (iii) ataque e defesa (como um jogo). Por se tratar de um modelo ainda não existente na prática, foi necessário pesquisar sobre possíveis formas de execução. A primeira questão a ser trabalhada quando se tem o uso de ML é o tratamento dos dados. Ferramentas inteligentes precisam de informação para análise e aprendizado.

A coleta de dados deve ser feita pelos bots (hosts infectados dentro do Simbo). Como o simulador ainda não contempla um ambiente de rede em sua totalidade (a nível de protocolos e serviços), decidiu-se inserir dados oriundos de várias bases de dados da internet. A ideia principal é trazer para dentro do Simbo um cenário semelhante ao que um botmaster encontra no mundo real. Portanto, cada host contém parte de um conjunto de eventos extraídos de bases públicas disponíveis na internet. Quando executado, o host carrega os dados e aplica filtros (previamente criados pelo C2) para encontrar as informações mais relevantes que serão enviadas para o C2. Assim, a primeira parte do projeto foi a divisão das bases.

A segunda parte desenvolvida trata-se de alterar o módulo de análise de dados para dar ao motor inteligente do C2 a possibilidade de trabalhar com os dados oriundos

dos bots. Após alguns testes, nós percebemos que a adição do motor dentro do Simbo apresentava problemas de processamento e precisava ser inserido em um módulo externo. Essa decisão se fez necessária porque ao elevar o número de bots na simulação a quantidade de dados sendo tratada tanto pelos bots quanto pelo C2 seria um problema para a escalabilidade do modelo. Assim, foi possível provar que a nova arquitetura do Simbo tem melhor desempenho que a anterior.

3.3. Controle online da simulação

Para permitir o funcionamento da nova arquitetura do Simbo foi desenvolvido o módulo de controle de simulação que faz a ponte entre os módulos dentro do OMNeT++ e os módulos externos (estação onde está instalado o OMNeT++). O modelo criado usa um arquivo ponte para indicar que um host específico está infectado ou que algum bot foi destruído e, portanto, não é mais possível enviar comandos a ele. É possível, ainda, enviar comandos para os bots disponíveis, emulando uma interface de controle de botnets (a Tabela 1 apresenta os comandos criados até aqui). Possibilita-se, assim, o envio de novas instruções para os bots pelo motor inteligente para além do que foi previamente programado. Para facilitar o uso deste módulo, foi desenvolvido um programa (ver Figura 5) que possibilita esse controle via linha de comando sem que seja necessário a edição manual do arquivo ponte.

Tabela 1. Comandos desenvolvidos para controlar a simulação

Comando	Descrição	Exemplo
infect host_name	Indica para a simulação que o host "host_name" (nome do host na simulação) foi infectado	infect host[0] - infecta o host "host[0]"
cure host_name	Indica para a simulação que o host "host_name" (nome do host na simulação) não está mais infectado	cure host[0] - remove a infecção do host "host[0]"
request host_name file	Requisita do "host_name", se ele estiver infectado, o arquivo "file"	request host[0] http.log - requisita do host "host[0]" o arquivo "http.log" filtrado

4. Simulando o primeiro ambiente para uma botnet inteligente

O primeiro cenário simulado de botnet com uso de técnicas de inteligência computacional é, basicamente, uma botnet com o motor inteligente na função do botmaster. Nesse protótipo, cada bot é responsável por capturar o tráfego da rede no qual o host infectado pertence, filtrá-lo e enviá-lo para o C2 que é administrado pelo motor inteligente.

Numa tentativa de simular esse tráfego de rede, foram distribuídas bases de dados obtidas da Web que continham dados de tráfego de rede. Como um simples exemplo, foram usadas as bases da MACCDC2010-2012 [MADCC]. O modelo funciona da seguinte forma: uma estação (IP) é escolhida como bot e, caso a base seja muito grande, escolhe-se aleatoriamente um subconjunto de IPs como bots (essa forma foi determinada por ter um número limitado de bases para simular grandes botnets). A ideia principal é obter um cenário semelhante ao que uma botnet opera. Cada bot é representado na simulação como host infectado.

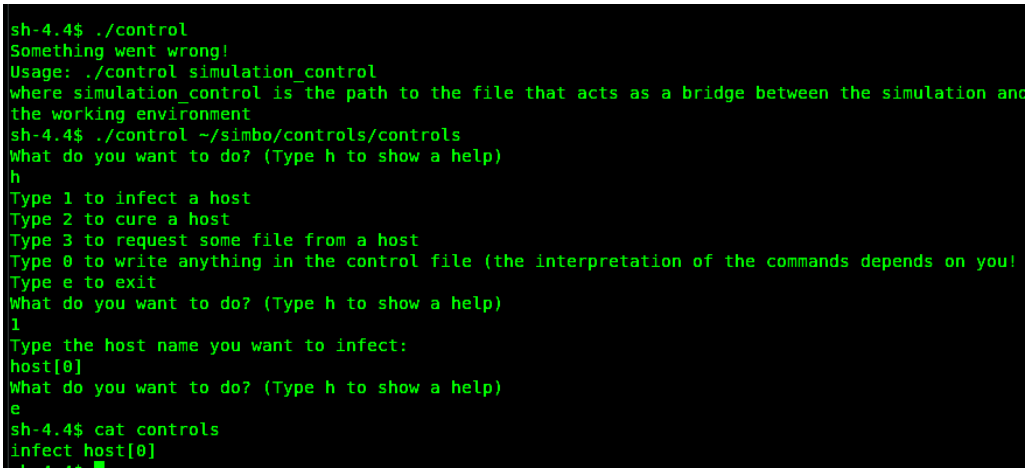

Figura 5. Exemplo de execução do software desenvolvido.

A fim de facilitar a filtragem de informações relevantes dessas bases, que estão no formato ”libpcap” , foi proposto um cenário em que os bots possuem uma ferramenta de análise de rede, como o *Bro Network Security Monitor* (Bro) [Paxson 1999] que foi contaminado e dominado pelo bot. Esse *framework* realiza o monitoramento da rede, produzindo arquivos de log facilmente interpretáveis.

Entretanto, os arquivos de log gerados pelo Bro são grandes e precisam ser filtrados. A tarefa de filtrar ou preprocessar os dados é muito importante dentro do contexto de ML. Por isso, primeiramente, foi necessária a análise de cada tipo de dado que o Bro insere nos logs seguindo os guias encontrados na página Web em [Bro] A partir do estudo descobrimos que o arquivo ”conn.log” é o principal arquivo de log, apresentando os eventos que ocorreram na rede e que fazem uso de conexões TCP, UDP e ICMP, contemplando a maior quantidade de informação sobre o ambiente do bot e foi usado como base de nossos filtros. A Tabela 2 mostra algumas das informações encontradas nesse arquivo. Como o objetivo do simulador é testar novos modelos de botnets, para criar os filtros precisamos entender as informações que mais podem interessar a um botmaster.

A Tabela 3 mostra os filtros construídos a partir das informações da Tabela 2 e um exemplo de informações filtradas levando em consideração duas questões importantes: (i) ambiente do bot (o host hospedeiro) e, (ii) ambiente externo (informações da rede onde está o host hospedeiro). Para alguns filtros foi necessário aplicar análise estatística, para outros apenas alguns comandos de agrupamento de dados. No final, cada bot possui uma tabela contemplando as informações filtradas. Essa tabela é enviada ao C2 que utilizará esses dados para alimentar o motor inteligente classificando o host infectado de acordo com suas possíveis vulnerabilidades e, por fim, o envio de comandos mais adequados ao bot. O desenvolvimento do motor inteligente está fora do escopo deste trabalho, mas está sendo tratado em um projeto paralelo.

Um problema detectado nessa fase foi que, como as bases de dados possuem tamanhos consideráveis, a tarefa de realizar a simulação sem qualquer paralelização se torna um trabalho pouco viável, se levarmos em consideração que a ideia é trabalhar com uma grande quantidade de hosts atuando na simulação. Simular o crescimento de uma botnet é essencial para qualquer estudo de novos modelos. Assim, decidimos distribuir a

Tabela 2. Campos do arquivo conn.log gerado pela ferramenta de monitoramento de rede Bro Network

Campo	Tipo	Descrição
ts	time	Timestamp do evento
uid	string	ID único da conexão
id.orig_h	addr	Endereço IP do ponto de origem do evento
id.orig_p	port	Porta TCP/UDP utilizada pelo ponto de origem do evento
id.resp_h	addr	Endereço IP do ponto de destino do evento
id.resp_p	port	Porta TCP/UDP utilizada pelo ponto de destino do evento
proto	transport_proto	Protocolo da camada de transporte utilizado
service	string	Protocolo da camada de aplicação utilizado
duration	interval	Intervalo decorrido entre o último pacote e o primeiro pacote que corresponde ao evento
orig_bytes	count	Tamanho do payload (em bytes) enviado pela origem do evento
resp_bytes	count	Tamanho do payload (em bytes) enviado pelo destino do evento
conn_state	string	Informa o estado da conexão (se foi bem sucedida, ou se ocorreu algum erro)
local_orig	bool	Verdadeiro se a conexão tiver origem local, falso caso contrário
missed_bytes	count	Quantidade de bytes perdidos
history	string	Histórico do estado da conexão
orig_pkts	count	Quantidade de pacotes enviados pelo ponto de origem
orig_ip_pkts	count	Tamanho (em bytes) dos pacotes enviados pelo ponto de origem
resp_pkts	count	Quantidade de pacotes enviados pelo ponto de destino
resp_ip_pkts	count	Tamanho (em bytes) dos pacotes enviados pelo ponto de destino
tunnel_parents	set	Se a conexão utilizar tunelamento, mostra os UID das conexões utilizadas no encapsulamento

simulação entre os processadores existentes na estação de trabalho para que pudéssemos alcançar maior capacidade de processamento e evitar problemas de escalabilidade.

5. Simulação Distribuída

Apesar do OMNeT++ já permitir a distribuição da simulação em processadores diferentes [Varga 2009], ele apresenta certas restrições, como por exemplo o uso de variáveis globais, não satisfeitas pelo INET, que não foi desenvolvido pensando em simulações distribuídas. Baseando-se no trabalho de Stoffer et al. [Stoffers et al. 2014] foram realizadas alterações no código-fonte das versões atuais do OMNeT++ e do INET de modo a permitir a simulação distribuída.

A Tabela 4 mostra uma comparação entre a versão sem paralelização e com paralelização do Simbo em um cenário que possui 10 sub-redes e 1000 hosts infectados que trocam mensagens simples com o C2 no intervalo de uma hora, e a Figura 6 mostra um gráfico *speedup* dos testes realizados. A versão com paralelização foi executada com 2, 3, 4, 5 e 6 processos, note que a divisão dos módulos entre os processos

Tabela 3. Filtros desenvolvidos para captura de informação e exemplos

Nome do filtro	Descrição	Exemplo de informação filtrada
bot_num_conn	Quantidade de eventos que utilizam conexões TCP, UDP e ICMP que o host infectado realizou.	15546
bot_orig_ports	Listagem das portas utilizadas pelo host infectado quando este faz uma requisição e que sejam menores que 1024, ou que igual a 8000 ou 8080.	8, 13
bot_resp_ports	Listagem das portas utilizadas pelo host infectado quando este responde uma requisição e que sejam maiores ou igual a 1024.	-
bot_protocols	Listagem dos protocolos da camada de transporte que o host infectado utilizou.	TCP, UDP, ICMP
bot_services	Listagem dos protocolos da camada de aplicação que o host infectado utilizou.	HTTP
bot_tx_pkts_rcv	Média dos pacotes recebidos pelo host infectado.	0.83
bot_tx_pkts_sent	Média dos pacotes enviados pelo host infectado.	1.21
bot_tx_pkts_total	Média dos pacotes recebidos e enviados pelo host infectado.	2.04
bot_tx_bytes_rcv	Média, em bytes, dos payloads recebidos pelo host infectado.	40.05
bot_tx_bytes_sent	Média, em bytes, dos payloads enviados pelo host infectado.	57.70
bot_tx_bytes_total	Média, em bytes, dos payloads recebidos e enviados pelo host infectado.	97.75
bot_tx_errors	Média de eventos do host infectado que apresentaram erros.	0.88
bot_tunneled	Verifica se o host infectado apresenta conexão que faça uso de tunelamento.	Falso
bot_proxy	Verifica se o host infectado é utilizado como proxy.	Falso

é feita manualmente, isto é, a simulação deve ser configurada previamente pelo usuário que deve selecionar os módulos que cada processo ficará responsável. Portanto, nos testes realizados, foram feitas divisões de modo a tentar aproveitar ao máximo os núcleos de processamento disponíveis. Cada cenário foi executado 100 vezes em uma máquina *quad-core*.

Como é possível notar, essa nova funcionalidade (paralelização) levou a uma diminuição no tempo de execução da simulação, apesar de um leve incremento no con-

Tabela 4. Resultados das simulações

Cenário	Consumo de memória (MiB)	Tempo de execução (s)
Sem paralelização	440.4 ± 3.3	112.8 ± 7.8
Com paralelização (2 processos)	444.6 ± 3.2	80.6 ± 5.9
Com paralelização (3 processos)	447.3 ± 2.8	47.6 ± 8.9
Com paralelização (4 processos)	453.1 ± 5.9	38.9 ± 5.8
Com paralelização (5 processos)	457.9 ± 1.9	38.4 ± 6.0
Com paralelização (6 processos)	461.0 ± 2.0	38.1 ± 10.1

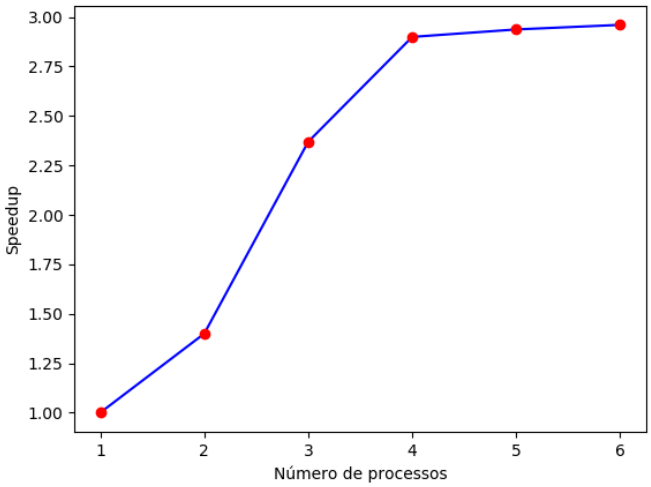

Figura 6. Gráfico Speedup dos testes realizados. Note a existência de saturação quando existem 5 processos sendo executados.

sumo de memória, o que já era esperado dado que alguns módulos tiveram que ser replicados nos diversos processos para que a simulação fosse executada. Observe que, devido a limitação do sistema, há saturação perto de cinco processos no *speedup*, o que também já era esperado. É possível que em uma máquina com mais núcleos de processamento o *speedup* sature com um valor maior e com um maior número de processos.

Portanto, a paralelização da simulação, com a diminuição do tempo de execução da simulação, levou a um aumento da escalabilidade do Simbo permitindo o aumento no número de hosts simulados. Além disso, a nova funcionalidade permite incrementar o processamento realizado dentro da simulação, isto é, as técnicas de inteligência computacional podem ser inseridas no ambiente de simulação, o que permitiria um C2 inteligente totalmente contido na simulação e, até mesmo, bots com ML embutido para levar ”inteligência”à ponta da botnet e transformar seu modus-operandi.

6. Conclusão e Trabalhos Futuros

Os resultados encontrados através das atividades desenvolvidas e apresentadas nesse trabalho demonstram que o Simbo já permite simular modelos de botnets que usam ML em seu C2 e até mesmo um modelo mais avançado que faça uso de ML nos bots. A adição de módulos externos ao OMNeT++ dentro do ambiente do Simbo retira o gargalo que poderia existir para situações de alta escalibilidade no número de bots e até mesmo

com a adição de mais de um C2 na mesma simulação. Embora possa existir gargalo na comunicação entre os módulos, em alguns testes executados não foi averiguada tal situação. Porém, é importante observar que testes com maior realidade somente serão possíveis com a simulação completa da botnet.

Também observamos que as adaptações no OMNeT++ e INET que permitem a execução distribuída da simulação levaram a uma diminuição no tempo de execução da simulação, o que possibilita um aumento no número de hosts presente na simulação, em comparação com a versão anterior que não possuía a opção de execução distribuída, ou, até mesmo, um incremento no volume de processamento que ocorre na simulação.

Como continuação desse trabalho, propõe-se a análise mais criteriosa da usabilidade do modelo de comunicação entre os módulos externos e o OMNeT++, testes de exaustão sobre a questão da paralelização do processamento para encontrar o ponto máximo permitido para grandes simulações dentro do Simbo e desenvolvimento do módulo de simulação de combate entre botnets e modelos de defesa.

Referências

Agarwal, S. (2010). *Performance analysis of peer-to-peer botnets using"The Storm Botnet"as an exemplar*. PhD thesis.

Athena. Código-fonte da botnet athena. `https://github.com/3val/Athena`. (acessado em 15/02/2018).

Balabanian, F., Danziger, M., and Henriques, M. A. A. (2016). Simbo - ambiente de simulação dedicado ao estudo de botnets. In *XVI Simpósio Brasileiro em Segurança da Informação e de Sistemas Computacionais. X Workshop de Trabalhos de Iniciação Científica e de Graduação*, pages 606–610.

Bro. Bro network log guides. acessado dia 25/06/2018.

Danziger, M. and Henriques, M. A. A. (2017). Attacking and defending with intelligent botnets. In *XXXV Simpósio Brasileiro de Telecomunicações e Processamento de Sinais - SBrT 2017*, pages 457–461.

INET (2017). Inet framework. `https://inet.omnetpp.org/`.

Kotenko, I. V. Experiments with simulation of botnets and defense agent teams.

MADCC. Mid-atlantic collegiate cyber defense competition. acessado dia 01/04/2018.

Paxson, V. (1999). Bro: a System for Detecting Network Intruders in Real-Time. *Computer Networks*, 31(23-24):2435–2463.

Stoffers, M., Bettermann, R., Gross, J., and Wehrle, K. (2014). Enabling distributed simulation of omnet++ INET models. *CoRR*, abs/1409.0994.

Varga, A. (2001). The omnet++ discrete event simulation system. In *In ESM'01*.

Varga, A. (2009). Parallel simulation made easy with omnet + +.