# EMULADOR DE TRACES DE BOT

*Carla Chrystina de Castro Pacheco Ferreira*, Ricardo Choren, Ronaldo M. Salles*
*Instituto Militar de Engenharia (IME) – Seção de Sistemas e Computação (SE/8).*
*Praça General Tibúrcio, 80, 22290-270, Praia Vermelha, Rio de Janeiro, RJ, Brasil.*
**carlapacheco@globo.com*

## RESUMO

A Internet esconde as atividades maliciosas de *botnets*. O tráfego de *bots* é difícil de ser obtido para fins de estudo acadêmico, pois pode conter informações sigilosas. O presente artigo propõe um emulador de *traces* de *bot* baseado nas etapas que mais perduram no ciclo de vida do *bot* que são a atualização e a manutenção de conectividade do *bot* à rede. Cada etapa é descrita e a sua implementação no emulador é detalhada. O resultado do experimento é apresentado e validado.

**Palavras-chave:** *Botnet*, *bot*, tráfego malicioso, emulação de tráfego.

## ABSTRACT

Internet hides malicious activities of botnets. Bot traffic is difficult to be obtained for academic studying purposes because it may contain sensitive information. This paper proposes a bot trace emulator based on bot lasting stages of it´s life cycle: upgrading and conectivity maintainance. Each stage is described and its implementation is detailed. The result of experiment is presented and validated.

**Keywords:** *Botnet*, *bot*, malicious traffic, traffic emulation.

## INTRODUÇÃO

Ataques distribuídos de negação de serviço (DDoS), fraudes eletrônicas, envios de *spam* (Tyagi & Aghila, 2011) em massa são exemplos de atividades maliciosas que ocorrem de maneira crescente na atualidade. Tais fatos despertam a atenção de governos, pesquisadores e entidades da área de Defesa Cibernética. Essas ações são articuladas através do uso de *botnets*, que são redes formadas por computadores infectados por *malwares* (códigos maliciosos) (Stone-Gross *et al*., 2009), que realizam funções predefinidas de maneira automatizada (Tyagi & Aghila, 2011), (Silva *et al*., 2012) e (Cooke *et al*., 2005). Estima-se que aproximadamente 20% dos computadores conectados à Internet participam de *botnets* (Silva

*et al*., 2012), (Sturgeon, 2007) e (Sadhan *et al*., 2009). Novas máquinas são infectadas diariamente, aumentamendo o exército de *bots* que formam a *botnet*.

Quando a máquina é infectada pelo *malware*, ela se torna um *bot*. Então, ocorre uma comunicação com o centro de Comando e Controle (C&C) para que o novo *bot* possa baixar a versão atualizada do *malware*, executar um ataque, propagar o *malware* na rede, fazer a manutenção de conectividade com a *botnet*, dentre outras atividades maliciosas. Essa comunicação deixa rastros de tráfego na rede (*traces*) que servem de insumo para ferramentas de análise e detecção de *botnets*.

Ao longo dos anos, as técnicas de evasão de detecção usadas pelas *botnets* foram adaptando e camuflando seu tráfego, tornando-o cada vez mais sutil para ser detectado. Como o tráfego de *botnet* está misturado a tráfegos reais, é possível que existam *traces* com atividades de *botnet* que contenham informações pessoais e/ou sigilosas (Silva *et al*., 2012). Por isso, *traces* reais com atividades de *botnets* com acesso público para fins de estudo são difíceis de serem obtidos (Abt, 2013). Isso dificulta o desenvolvimento e a validação de novas técnicas de detecção de *botnets* (Silva *et al*., 2012) e (Tyagi & Aghila, 2011).

O DARPA (*Defense Advanced Research Projects Agency*) é uma referência mundial de *traces* com atividades maliciosas, porém são antigos (1999-2000). A atualização deste banco não acompanhou a evolução das técnicas e características das *botnets*. Em 2011 e 2012, aproximadamente 76% dos pesquisadores das linhas de Segurança Cibernética construíram manualmente os dados de tráfego para seus trabalhos (Abt, 2013). Dos *traces* disponibilizados para pesquisa, a duração da atividade maliciosa é curta e sua coleta, pontual. A visão conjunta desta atividade em vários pontos da rede e com acesso público não foi encontrada até o momento. Portanto, os poucos *traces* disponibilizados para pesquisa apresentam problemas nos fatores qualidade, representatividade e contemporaneidade.

A área de validação de ferramentas de identificação, detecção e mitigação de *botnets* enfrenta o problema de carência desses *traces* (Silva *et al.*, 2012 e Lee, 2009). Esses tráfegos servem de referência em auditorias, perícias, métricas, testes de ferramentas que lidam com tráfego de rede, dentre outros.

Para resolver esta questão, foram feitas propostas de ferramentas de geração de tráfego sintético de *botnet* que, além de escassas, não são sempre acessíveis (Jackson *et al*., 2009 e Lee, 2009). Elas ainda esbarram em limitações técnicas, como a construção de ambientes mais próximos à realidade. A execução do *malware* em máquinas virtuais, que serveriam como uma “caixa de vidro” para observação do comportamento do *malware*, tanto para atividades internas e externas à rede, esbarra em técnicas de identificação dessa observação por parte do *malware*. Essas técnicas mudam o comportamento do *malware* (Lee, 2009), anulando sua atividade maliciosa para não serem observadas nem rastreadas.

Devido à carência de *traces* de *bot* para fins de estudo acadêmico na área de Segurança da Informação, o presente artigo propõe um emulador de *traces* de *bot*. Para isso, foi feito um estudo de modelos de comunicação de *bot*, sendo o foco a área de comunicação entre o *bot* e o C&C, o comportamento observável na rede, que tem como consequência o *trace* gerado. O *bot* é a unidade mínima da *botnet*, por isso ele foi escolhido como o ponto de partida para análise. A arquitetura centra-

lizada (cliente-servidor) foi escolhida por existirem diversas *botnets* ainda fazendo seu uso atualmente e também por ser a mais escalável para fins de experimentação.

A contribuição deste artigo é o emulador de *traces* de *bot* para geração de tráfego de *bot* na arquitetura centralizada que gera tráfego próximo do real. A vantagem do emulador perante à instanciação de um *bot* real é que o pesquisador fica isento de parcipação de um ataque distribuído de negação de serviço num cenário real, além do envolvimento em outras atividades maliciosas. Isso poderia acontecer caso sua máquina se tornasse um *bot* real (Zou & Cunningham, 2006).

## OBJETIVO DO TRABALHO

O emulador proposto será executado em um ambiente real, ou seja, o tráfego gerado será misturado naturalmente ao tráfego de outras aplicações de rede, preservando a temporização dos tráfegos do *bot* em relação a outros tráfegos de rede. Isso servirá de insumo para o desenvolvimento de técnicas de detecção e mitigação de *botnets* que focam na atividade e no comportamento do *bot* na rede.

A qualidade do tráfego gerado será comprovada através de experimentos que comparam a sua fidedignidade a artigos bem descritivos do comportamento dos *bots*. Sendo assim, ao final será mostrada uma validação qualitativa.

A emulação de um *bot* em um ambiente real permitirá aos pesquisadores fazerem uso desta ferramenta para controle do *bot* emulado que desejam instanciar. Isso poderá ser feito sem a violação de normas de segurança, pois o emulador do *bot* será inteiramente configurado por seu pesquisador.

## CONCEITOS SOBRE *BOTNETS*

Define-se *bot* como um termo genérico derivado de "*ro-Bot*" que é utilizado para descrever um script ou um conjunto de scripts projetados para desempenhar funções predefinidas de maneira automatizada (Tyagi & Aghila, 2011). No contexto deste artigo, as funções predefinidas são de cunho malicioso.

Dando prosseguimento a esta ideia, define-se *botnet* como redes formadas por computadores infectados por *malwares* que realizam funções predefinidas de forma automatizada, o que é uma das grandes ameaças na atualidade (Silva *et al*., 2012; Tyagi & Aghila, 2011; Cooke *et al*., 2005; Choi *et al*., 2009; e Ceron *et al.*, 2010).

Cibercrimes e ciberataques são realizados através das *botnets* que atuam como um exército particular do *botmaster* (detentor da *botnet*). No submundo do crime, as *botnets* podem ser alugadas por quantidade de *bots*, tempo e serviço determinados (Silva *et al*., 2012; Goncharov, 2012).

Há um consenso na literatura sobre a classificação das *botnets* no tocante ao tipo de arquitetura utilizada, pondendo ser centralizada (C&C) ou distribuída (P2P), (Silva *et al*., 2012; Lee, 2009; Feily *et al*., 2009; Yong *et al*., 2012). No tipo centralizada, o *botmaster* distribui o tráfego da *botnet* (comandos de ataque, atualizações,

informações) em pontos chave, denominados centros de comando e controle, ou C&Cs, que se comunicam com o *bot* em intervalos de tempo previamente definidos. Os fluxos de comunicação são do *bot* para C&C e do C&C para o *bot*, apenas. Já na arquitetura descentralizada, o fluxo de comunicação pode ser distribuído entre os nós (*peers*) que podem atuar como C&Cs, distribuindo comandos, dependendo da característica da *botnet*. A diferença na arquitetura descentralizada está na comunicação entre *bots* diretamente, o que não ocorre na centralizada. Em ambas, há a comunicação entre C&C e *bot*.

A máquina é contaminada por um *malware* que pode ser espalhado na *Internet*, ou noutros meios, através de propagação própria, visando captar mais e mais *bots*. A partir daí, o *bot* realiza atividades de manutenção de conectividade com a *botnet*, atualizações do *malware*, ataques e propagação. As duas últimas são as que geram um maior volume de tráfego, apesar de perdurarem um menor período de tempo no ciclo de vida do *bot*, se comparadas com as outras. Mesmo assim, como essas são as atividades que causam danos mais diretos aos serviços, os trabalhos na literatura da área de atividades maliciosas são mais concentrados em ataque e propagação.

## ARQUITETURA CENTRALIZADA

Neste tipo de arquitetura, quando o *bot* é infectado, ocorre sua conexão com a *botnet* através dos C&Cs. Estes centros também podem atuar como servidores de atualização do *malware*. Esse cenário é similar ao clássico modelo de rede cliente-servidor (Silva *et al.*, 2012), no qual os *bots* fazem requisições a um servidor central (Wanga *et al.*, 2010). O *botmaster* controla diretamente a *botnet* (Gu *et al.*, 2008) através dos C&Cs que possuem um canal de comunicação relativamente estável. Com isso, cada C&C se torna um ponto fraco desta arquitetura (Silva *et al.*, 2012) por ser um concentrador de fluxos de comando.

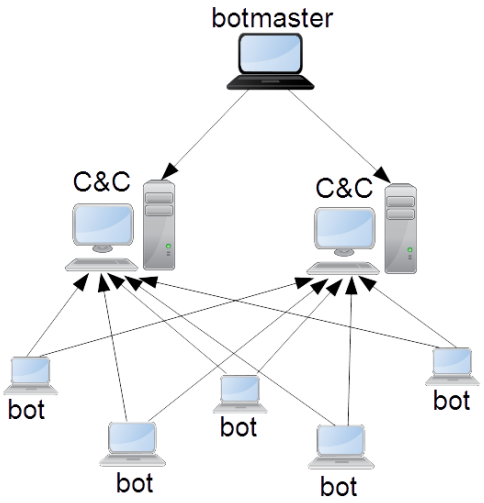



**Figura 1.** *Arquitetura Centralizada*

Na arquitetura centralizada (Figura 1), um *bot* não tem conhecimento sobre os outros *bots* da rede (Stone-Gross *et al.*, 2009). Existe, apenas, uma hierarquia de comando

dos servidores C&C para os *bots*. A comunicação entre o C&C e os *bots* inclui o envio de comandos e recebimento de respostas que podem ser entendidas como relatórios do resultado ou do progresso de uma tarefa executada pelo *bot* (Gu *et al.*, 2008). O alto sincronismo na comunicação com baixo tempo de resposta é um traço marcante desta arquitetura (Silva *et al*., 2012; Gu *et al*., 2008; Sadhan *et al*., 2009).

Há dois mecanismos de comunicação entre *bots* e servidores C&C: *pull* e push. No mecanismo *push*, o C&C envia comandos ao *bots*, enquanto que no mecanismo *pull*, os *bots* fazem a requisição dos comandos ao C&C. O funcionamento da *botnet* que implementa o mecanismo *pull* se dá da seguinte forma: o *botmaster* injeta um comando num arquivo no servidor C&C. Periodicamente, os *bots* se conectam a este servidor para verificar novos comandos. No caso do protocolo HTTP, os *bots* se conectam através de uma requisição a uma URL neste servidor e recebem o comando como resposta. No mecanismo *pull*, o *botmaster* não possui um controle em tempo real como no mecanismo *push*, mas ainda mantém o controle rígido da *botnet*. Já o mecanismo *push* é mais característico de *botnets* que utilizam o protocolo IRC. Os *bots* se conectam a servidores C&C e aguardam por comandos enviados pelo *botmaster* a estes servidores. Quando qualquer comando é recebido pelo C&C, este é repassado a todos os *bots* conectados a ele. Com isso, o *botmaster* possui o controle da *botnet* em tempo real (Gu *et al*., 2008).

Vale ressaltar que de acordo com a análise do perfil predominante de comunicação de um *bot* é que é determinado se o mesmo segue o mecanismo *pull* ou *push*. Ambos os mecanismos podem existir em diversos fluxos de comunicação entre os *bots* e os servidores C&C, mas um deles é que é predominante, no geral.

Os *bots* ainda possuem um comportamento espacial-temporal correlacionado, como, por exemplo, janelas de tempo similares. Isso é definido de acordo com as atividades de respostas pré-programadas do *bot* a comandos recebidos. Todo comportamento do *bot* está predefinido no *malware*. Quando os *bots* possuem o mesmo perfil de atividade de tráfego ou enviam mensagens semelhantes, esse comportamento recorrente é percebido na rede e se mantém por todo o ciclo de vida do *bot*.

Os estudos de (Gu *et al*., 2008) e (Sadhan *et al*., 2009) relatam que o comportamento recorrente de respostas similares de clientes enviadas em uma janela de tempo semelhante não ocorre em um tráfego sem atividade maliciosa na rede.

O *bot* recém-infectado necessita se comunicar com um ou mais servidores C&C para fins de atualização da versão do *malware* e da lista de servidores com os quais ele irá se comunicar. Esta lista de servidores C&C pode ser baixada juntamente com o *malware* ou adquirida em posteriores requisições a um C&C. A busca por esta lista é periódica para que o *bot* se mantenha atualizado e conectado à *botnet*. Portanto, este comportamento de periodicidade de conexão é comum para qualquer *bot*. Sendo assim, o fator tempo (ou a temporização) deve ser considerado no tráfego de um *bot*.

## MODELOS DE COMUNICAÇÃO DE *BOTNET*

A literatura científica da área de Segurança da Informação foi tomada como base para a extração de características gerais de modelo de comunicação de *botnet*. O foco do emulador é a produção de *traces*, ou seja, o que é visível na rede.

Portanto, apenas as características que impactam na rede foram levadas em consideração nesta análise. Atividades de mudanças internas na máquina, como modificação de registros do Sistema Operacional, não foram consideradas para análise. Dentre os trabalhos observados, os que se destacam embasam as características genéricas do emulador.

Os trabalhos de (Silva *et al*., 2012), (Tyagi & Aguila, 2011) e (Feily *et al*., 2009) contém pesquisas abrangentes feitas com a construção de uma espécie de catálogo das *botnets* ao longo do tempo. Por conta disso, esses três trabalhos foram destacados.

Em (Silva *et al*., 2012), baseado em (Feily *et al*., 2009) e (Zhu *et al*., 2008), é observada a seguinte estrutura:

- *Initial Infection*: infecção inicial da máquina para se tornar um *bot*.
- *Secondary injection*: procura e *download* de *malware*. Nesta etapa, pode ser incluído o *download* de arquivos adicionais de configuração da *botnet*.
- *Connection or Rally*: estabelecimento de conexão com troca de mensagens com o C&C para manutenção da comunicação.
- *Malicious Activities*: realização de atividades maliciosas, o que inclui o ataque e a propagação.
- *Maintenance and upgrading*: atualizações e troca de mensagens entre o *bot* e o C&C, com o intuito do *bot* continuar participando da *botnet*.

Os estudos de (Tyagi & Aguila, 2011) e (Feily *et al*., 2009) propõem estruturas parecidas entre si. Em (Feily *et al*., 2009), é destacada uma sequência de atividades, na ordem: infecção inicial, injeção secundária, conexão, atividades maliciosas vindas do C&C, manutenção e atualização. Tanto as atividades maliciosas do C&C quanto a manutenção e a atualização contam com a participação do *botmaster*.

As etapas de atualização e manutenção de conectividade do *bot* são comuns a todos os trabalhos apresentados. A atualização corresponde a *Maintenance and upgrading* do trabalho de (Silva *et al*., 2012) e às atividades de atualização dos trabalhos (Tyagi & Aguila, 2011) e (Feily *et al*., 2009). Já a manutenção de conectividade do *bot* corresponde a "*Connection or Rally"* e a "*Maintenance and upgrading"* de (Silva *et al*., 2012) com as atividades de obtenção de possíveis atualizações da lista de C&C que pode ser enviada separadamente do *malware* e atividades de conectividade do *bot* à *botnet*. As atividades de manutenção também são análogas às atividades de manutenção dos trabalhos de (Tyagi & Aguila, 2011) e (Feily *et al*., 2009).

Além de estarem em todos os modelos de comunicação apresentados, ambas as etapas são iterativas e perduram um maior tempo no ciclo de vida do *bot.* Isso pode ser destacado como um traço característico do comportamento básico do *bot* na rede.

## EMULADOR

O emulador proposto foi dirigido para atender aos seguintes pontos: comportamento do tráfego do *bot* na rede, características gerais da arquitetura centralizada, protocolo HTTP, mecanismo *pull*, etapas de atualização e manutenção de conectividade do *bot*, extensibilidade para adição de demais etapas (como ataque

e propagação) e customização das atividades e funcionalidades.

As *botnets* que fazem uso do protocolo HTTP são baseadas no mecanismo *pull* de conexão (Gu *et al.*, 2008; Zeidanloo & Manaf, 2009). Por isso, no emulador proposto, a busca por comandos está inicialmente implementada para partir dos *bots* para os servidores C&C.

As características mais comuns de comunicação do *bot* com os servidores C&Cs, assim como as etapas de atualização e manutenção de conectividade do *bot*, estão implementadas e podem ser customizadas no emulador. As requisições do *bot* aos servidores C&C podem ter sua temporização definida, assim como o conteúdo das mensagens e seus cabeçalhos. Outras etapas e atividades de comunicação do *bot* podem ser adicionadas em pontos específicos de extensão do emulador, como as de ataque e propagação que fazem parte de *Atividades Maliciosas* (terceira etapa de Silva *et al.*, 2012). Num cenário real de uma *botnet* sendo executada, essas atividades podem ocorrer a qualquer momento, dependendo do comando do *botmaster* (em casos de ataque DoS, por exemplo) e de acordo com o que está pré-programado no *malware* (o caso da propagação). No caso específico da propagação, esta atividade pode ser realizada paralelamente com outras já implementadas no emulador. O modo de execução das etapas e atividades pode ser sequencial ou em paralelo, de acordo com o que o pesquisador programar no emulador. Com isso, os modelos existentes de ciclo de vida do *bot* são atendidos pelo emulador.

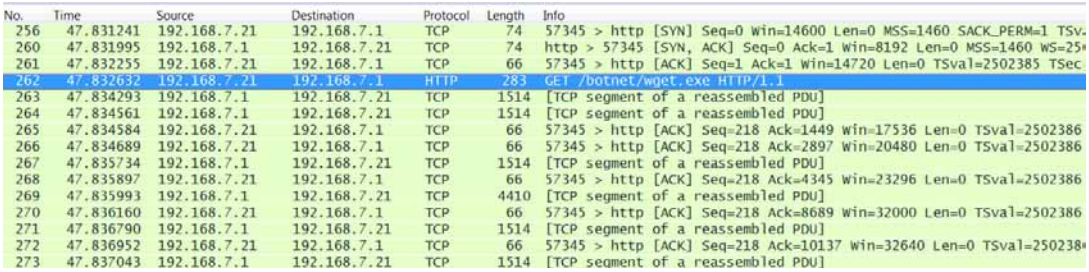
Uma característica bastante comum nas *botnets* é o uso de algoritmos de geração de domínio dinâmicos (DGAs) que produzem numerosos nomes de domínios periodicamente. Para atender a esta demanda, o emulador possui um ponto de extensão antes de cada requisição a ser feita pelo *bot*. Isso permite a configuração de outra requisição a um determinado domínio, seja para checagem de sua existência (no caso de domínios inválidos), seja para checagem de conectividade com a rede (em casos do *bot* não estar conectado a nenhuma rede). A checagem de servidor, a princípio, está implementada para utilizar o método “GET” e é feita com uma requisição somente para o nome do domínio, sem a complementação do caminho completo do arquivo no servidor. Já a requisição a um determinado arquivo é feita utilizando a URL completa.

Algumas características básicas do emulador incluem o comportamento do envio e do recebimento de mensagens. Os cenários de sucesso (código de retorno “200 OK”) estão implementados, já os de exceção e maiores desdobramentos do cenário de sucesso (como interpretação de comandos enviados na resposta) são contemplados em métodos de tratamento de comando que são disparados após o recebimento de uma mensagem. Este método engloba o tempo esperado pelo *bot*, o que emula uma execução do comando recebido. Os métodos de tratamento de comando estão pontualmente declarados no código e podem ser modificados ou estendidos (o comportamental do *bot* fica de acordo com o que for instanciado).

**Etapa de Atualização**

O *download* do *malware* (Figura 2) deve ser bem-sucedido para a manutenção de conectividade do *bot* à *botnet*. Uma opção, no caso desta atividade, é o envio de uma mensagem de sucesso de *download* do *malware* ao C&C ao fim do

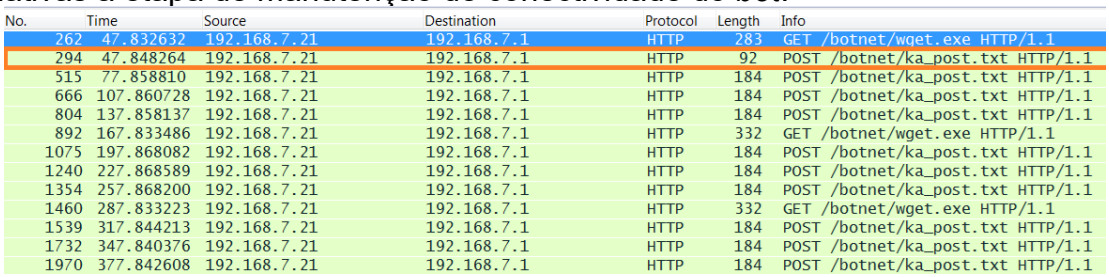
*download* bem-sucedido do *malware*. Pode ser percebido que a requisição para *download* do *malware* é diferente da requisição para envio de mensagem de sucesso desta atividade. Portanto, tratamentos distintos destas mensagens devem ser feitos. Mensagens de sucesso de recebimento de respostas e duplas requisições possuem tratamento e implementação distintos, visando abranger diversos tipos de comportamentos em requisições e respostas com objetivos diferentes. Método de envio, cabeçalho e conteúdo das mensagens podem ser modificados. O emulador atende a esta demanda.



| No. | Time | Source | Destination | Protocol | Length | Info |
|---|---|---|---|---|---|---|
| 256 | 47.831241 | 192.168.7.21 | 192.168.7.1 | TCP | 74 | 57345 > http [SYN] Seq=0 Win=14600 Len=0 MSS=1460 SACK_PERM=1 TSv. |
| 260 | 47.831995 | 192.168.7.1 | 192.168.7.21 | TCP | 74 | http > 57345 [SYN, ACK] Seq=0 Ack=1 Win=8192 Len=0 MSS=1460 WS=25 |
| 261 | 47.832255 | 192.168.7.21 | 192.168.7.1 | TCP | 66 | 57345 > http [ACK] Seq=1 Ack=1 Win=14720 Len=0 TSval=2502385 TSec |
| 262 | 47.832632 | 192.168.7.21 | 192.168.7.1 | HTTP | 283 | GET /botnet/wget.exe HTTP/1.1 |
| 263 | 47.834293 | 192.168.7.1 | 192.168.7.21 | TCP | 1514 | [TCP segment of a reassembled PDU] |
| 264 | 47.834561 | 192.168.7.1 | 192.168.7.21 | TCP | 1514 | [TCP segment of a reassembled PDU] |
| 265 | 47.834584 | 192.168.7.21 | 192.168.7.1 | TCP | 66 | 57345 > http [ACK] Seq=218 Ack=1449 Win=17536 Len=0 TSval=2502386 |
| 266 | 47.834689 | 192.168.7.21 | 192.168.7.1 | TCP | 66 | 57345 > http [ACK] Seq=218 Ack=2897 Win=20480 Len=0 TSval=2502386 |
| 267 | 47.835734 | 192.168.7.1 | 192.168.7.21 | TCP | 1514 | [TCP segment of a reassembled PDU] |
| 268 | 47.835897 | 192.168.7.21 | 192.168.7.1 | TCP | 66 | 57345 > http [ACK] Seq=218 Ack=4345 Win=23296 Len=0 TSval=2502386 |
| 269 | 47.835993 | 192.168.7.1 | 192.168.7.21 | TCP | 4410 | [TCP segment of a reassembled PDU] |
| 270 | 47.836160 | 192.168.7.21 | 192.168.7.1 | TCP | 66 | 57345 > http [ACK] Seq=218 Ack=8689 Win=32000 Len=0 TSval=2502386 |
| 271 | 47.836790 | 192.168.7.1 | 192.168.7.21 | TCP | 1514 | [TCP segment of a reassembled PDU] |
| 272 | 47.836952 | 192.168.7.21 | 192.168.7.1 | TCP | 66 | 57345 > http [ACK] Seq=218 Ack=10137 Win=32640 Len=0 TSval=250238 |
| 273 | 47.837043 | 192.168.7.1 | 192.168.7.21 | TCP | 1514 | [TCP segment of a reassembled PDU] |

**Figura 2.** *Download do arquivo de atualização do malware (/botnet/wget.exe)*

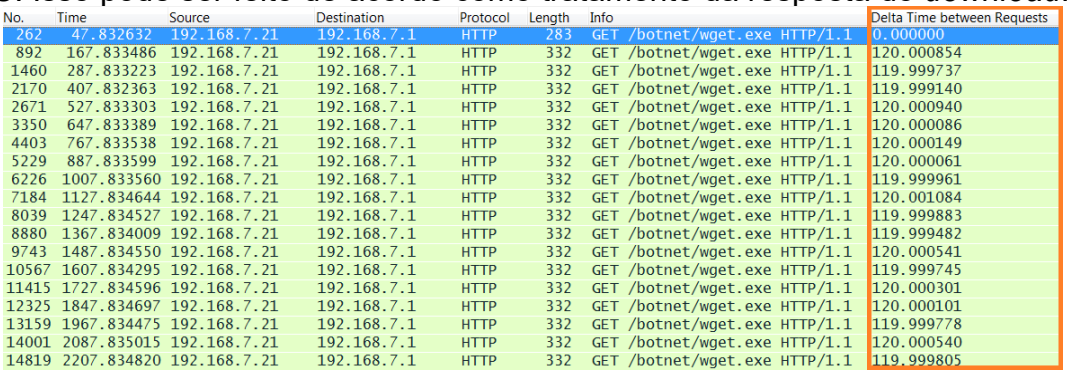
Na Figura 3, pode-se notar que há quatro mensagens com o método POST entre a primeira e a segunda requisições com o método GET. A primeira mensagem com o método POST é a de sucesso de *download*, enquanto que as demais são relativas à etapa de manutenção de conectividade do *bot*.



| No. | Time | Source | Destination | Protocol | Length | Info |
|---|---|---|---|---|---|---|
| 262 | 47.832632 | 192.168.7.21 | 192.168.7.1 | HTTP | 283 | GET /botnet/wget.exe HTTP/1.1 |
| 294 | 47.848264 | 192.168.7.21 | 192.168.7.1 | HTTP | 92 | POST /botnet/ka_post.txt HTTP/1.1 |
| 515 | 77.858810 | 192.168.7.21 | 192.168.7.1 | HTTP | 184 | POST /botnet/ka_post.txt HTTP/1.1 |
| 666 | 107.860728 | 192.168.7.21 | 192.168.7.1 | HTTP | 184 | POST /botnet/ka_post.txt HTTP/1.1 |
| 804 | 137.858137 | 192.168.7.21 | 192.168.7.1 | HTTP | 184 | POST /botnet/ka_post.txt HTTP/1.1 |
| 892 | 167.833486 | 192.168.7.21 | 192.168.7.1 | HTTP | 332 | GET /botnet/wget.exe HTTP/1.1 |
| 1075 | 197.868082 | 192.168.7.21 | 192.168.7.1 | HTTP | 184 | POST /botnet/ka_post.txt HTTP/1.1 |
| 1240 | 227.868589 | 192.168.7.21 | 192.168.7.1 | HTTP | 184 | POST /botnet/ka_post.txt HTTP/1.1 |
| 1354 | 257.868200 | 192.168.7.21 | 192.168.7.1 | HTTP | 184 | POST /botnet/ka_post.txt HTTP/1.1 |
| 1460 | 287.833223 | 192.168.7.21 | 192.168.7.1 | HTTP | 332 | GET /botnet/wget.exe HTTP/1.1 |
| 1539 | 317.844213 | 192.168.7.21 | 192.168.7.1 | HTTP | 184 | POST /botnet/ka_post.txt HTTP/1.1 |
| 1732 | 347.840376 | 192.168.7.21 | 192.168.7.1 | HTTP | 184 | POST /botnet/ka_post.txt HTTP/1.1 |
| 1970 | 377.842608 | 192.168.7.21 | 192.168.7.1 | HTTP | 184 | POST /botnet/ka_post.txt HTTP/1.1 |

*Figura 3. Mensagem de sucesso de download*

A etapa de atualização é iterativa e ocorre de acordo com um tempo predefinido no arquivo de configuração. No exemplo da Figura 4, a temporização entre as requisições foi configurada para 120 segundos. Após a requisição característica desta etapa, uma mensagem de sucesso de *download* pode ser enviada para o C&C. Isso pode ser feito de acordo como tratamento da resposta do *download*.



| No. | Time | Source | Destination | Protocol | Length | Info | Delta Time between Requests |
|---|---|---|---|---|---|---|---|
| 262 | 47.832632 | 192.168.7.21 | 192.168.7.1 | HTTP | 283 | GET /botnet/wget.exe HTTP/1.1 | 0.000000 |
| 892 | 167.833486 | 192.168.7.21 | 192.168.7.1 | HTTP | 332 | GET /botnet/wget.exe HTTP/1.1 | 120.000854 |
| 1460 | 287.833223 | 192.168.7.21 | 192.168.7.1 | HTTP | 332 | GET /botnet/wget.exe HTTP/1.1 | 119.999737 |
| 2170 | 407.832363 | 192.168.7.21 | 192.168.7.1 | HTTP | 332 | GET /botnet/wget.exe HTTP/1.1 | 119.999140 |
| 2671 | 527.833303 | 192.168.7.21 | 192.168.7.1 | HTTP | 332 | GET /botnet/wget.exe HTTP/1.1 | 120.000940 |
| 3350 | 647.833389 | 192.168.7.21 | 192.168.7.1 | HTTP | 332 | GET /botnet/wget.exe HTTP/1.1 | 120.000086 |
| 4403 | 767.833538 | 192.168.7.21 | 192.168.7.1 | HTTP | 332 | GET /botnet/wget.exe HTTP/1.1 | 120.000149 |
| 5229 | 887.833599 | 192.168.7.21 | 192.168.7.1 | HTTP | 332 | GET /botnet/wget.exe HTTP/1.1 | 120.000061 |
| 6226 | 1007.833560 | 192.168.7.21 | 192.168.7.1 | HTTP | 332 | GET /botnet/wget.exe HTTP/1.1 | 119.999961 |
| 7184 | 1127.834644 | 192.168.7.21 | 192.168.7.1 | HTTP | 332 | GET /botnet/wget.exe HTTP/1.1 | 120.001084 |
| 8039 | 1247.834527 | 192.168.7.21 | 192.168.7.1 | HTTP | 332 | GET /botnet/wget.exe HTTP/1.1 | 119.999883 |
| 8880 | 1367.834009 | 192.168.7.21 | 192.168.7.1 | HTTP | 332 | GET /botnet/wget.exe HTTP/1.1 | 119.999482 |
| 9743 | 1487.834550 | 192.168.7.21 | 192.168.7.1 | HTTP | 332 | GET /botnet/wget.exe HTTP/1.1 | 120.000541 |
| 10567 | 1607.834295 | 192.168.7.21 | 192.168.7.1 | HTTP | 332 | GET /botnet/wget.exe HTTP/1.1 | 119.999745 |
| 11415 | 1727.834596 | 192.168.7.21 | 192.168.7.1 | HTTP | 332 | GET /botnet/wget.exe HTTP/1.1 | 120.000301 |
| 12325 | 1847.834697 | 192.168.7.21 | 192.168.7.1 | HTTP | 332 | GET /botnet/wget.exe HTTP/1.1 | 120.000101 |
| 13159 | 1967.834475 | 192.168.7.21 | 192.168.7.1 | HTTP | 332 | GET /botnet/wget.exe HTTP/1.1 | 119.999778 |
| 14001 | 2087.835015 | 192.168.7.21 | 192.168.7.1 | HTTP | 332 | GET /botnet/wget.exe HTTP/1.1 | 120.000540 |
| 14819 | 2207.834820 | 192.168.7.21 | 192.168.7.1 | HTTP | 332 | GET /botnet/wget.exe HTTP/1.1 | 119.999805 |

**Figura 4.** *Temporização das mensagens de atualização*

### Etapa de Manutenção de Conectividade

Da mesma maneira, na etapa de manutenção de conectividade do *bot*, podem

ser configurados os dois fluxos iterativos existentes nesta etapa. Um fluxo é para envio de dados do *bot* ao C&C (geralmente com o método POST, mas também pode ser utilizado o método GET, no caso da *botnet* não utilizar o método POST) e o outro é para checagem de conectividade do *bot* à rede (geralmente com o método GET). Ambos os fluxos podem coexistir (Figura 5). Quando isto ocorre, todas as requisições são feitas de modo paralelo (intervalo de tempo entre as requisições de método GET e POST é próximo de zero), independente do outro fluxo. Apenas quando as atividades de atualização estiverem no tempo de serem executadas, todas as atividades da etapa de manutenção de conectividade do *bot* são suspensas e retomadas após o término da etapa de atualização.

| No. | Time | Source | Destination | Protocol | Length | Info | Delta Time between Requests |
|---|---|---|---|---|---|---|---|
| 7384 | 1157.840397 | 192.168.7.21 | 192.168.7.1 | HTTP | 278 | GET /botnet/ka_get.html HTTP/1.1 | 60.000770 |
| 7388 | 1157.841304 | 192.168.7.21 | 192.168.7.1 | HTTP | 184 | POST /botnet/ka_post.txt HTTP/1.1 | 0.000907 |
| 7587 | 1187.840605 | 192.168.7.21 | 192.168.7.1 | HTTP | 278 | GET /botnet/ka_get.html HTTP/1.1 | 29.999301 |
| 7590 | 1187.841185 | 192.168.7.21 | 192.168.7.1 | HTTP | 184 | POST /botnet/ka_post.txt HTTP/1.1 | 0.000580 |
| 7833 | 1217.841321 | 192.168.7.21 | 192.168.7.1 | HTTP | 278 | GET /botnet/ka_get.html HTTP/1.1 | 30.000136 |
| 7835 | 1217.841666 | 192.168.7.21 | 192.168.7.1 | HTTP | 184 | POST /botnet/ka_post.txt HTTP/1.1 | 0.000345 |
| 8224 | 1277.841190 | 192.168.7.21 | 192.168.7.1 | HTTP | 184 | POST /botnet/ka_post.txt HTTP/1.1 | 59.999524 |
| 8231 | 1277.841987 | 192.168.7.21 | 192.168.7.1 | HTTP | 278 | GET /botnet/ka_get.html HTTP/1.1 | 0.000797 |
| 8429 | 1307.841637 | 192.168.7.21 | 192.168.7.1 | HTTP | 184 | POST /botnet/ka_post.txt HTTP/1.1 | 29.999650 |
| 8435 | 1307.842386 | 192.168.7.21 | 192.168.7.1 | HTTP | 278 | GET /botnet/ka_get.html HTTP/1.1 | 0.000749 |
| 8668 | 1337.840647 | 192.168.7.21 | 192.168.7.1 | HTTP | 184 | POST /botnet/ka_post.txt HTTP/1.1 | 29.998261 |

**Figura 5.** *Temporização das requisições da etapa de Manutenção de Conectividade*

Os temporizadores das requisições com o método GET e POST da etapa de manutenção de conectividade do *bot* foram configurados para 30 segundos cada. Isso justifica o intervalo de tempo ficar próximo de 60 segundos a cada 4 requisições com o mesmo método (Figura 6), pois o temporizador da etapa de atualização foi definido para 120 segundos. O tráfego referente à etapa de atualização coexiste com o tráfego da etapa de manutenção de conectividade, mas apenas o último é mostrado nas figuras 5 e 6. O emulador possui geração contínua de tráfego, portanto, essas atividades ocorrem indefinidamente.

| No. | Time | Source | Destination | Protocol | Length | Info | Delta Time between Requests |
|---|---|---|---|---|---|---|---|
| 7384 | 1157.840397 | 192.168.7.21 | 192.168.7.1 | HTTP | 278 | GET /botnet/ka_get.html HTTP/1.1 | 60.000770 |
| 7388 | 1157.841304 | 192.168.7.21 | 192.168.7.1 | HTTP | 184 | POST /botnet/ka_post.txt HTTP/1.1 | 0.000907 |
| 7587 | 1187.840605 | 192.168.7.21 | 192.168.7.1 | HTTP | 278 | GET /botnet/ka_get.html HTTP/1.1 | 29.999301 |
| 7590 | 1187.841185 | 192.168.7.21 | 192.168.7.1 | HTTP | 184 | POST /botnet/ka_post.txt HTTP/1.1 | 0.000580 |
| 7833 | 1217.841321 | 192.168.7.21 | 192.168.7.1 | HTTP | 278 | GET /botnet/ka_get.html HTTP/1.1 | 30.000136 |
| 7835 | 1217.841666 | 192.168.7.21 | 192.168.7.1 | HTTP | 184 | POST /botnet/ka_post.txt HTTP/1.1 | 0.000345 |
| 8224 | 1277.841190 | 192.168.7.21 | 192.168.7.1 | HTTP | 184 | POST /botnet/ka_post.txt HTTP/1.1 | 59.999524 |
| 8231 | 1277.841987 | 192.168.7.21 | 192.168.7.1 | HTTP | 278 | GET /botnet/ka_get.html HTTP/1.1 | 0.000797 |
| 8429 | 1307.841637 | 192.168.7.21 | 192.168.7.1 | HTTP | 184 | POST /botnet/ka_post.txt HTTP/1.1 | 29.999650 |
| 8435 | 1307.842386 | 192.168.7.21 | 192.168.7.1 | HTTP | 278 | GET /botnet/ka_get.html HTTP/1.1 | 0.000749 |
| 8668 | 1337.840647 | 192.168.7.21 | 192.168.7.1 | HTTP | 184 | POST /botnet/ka_post.txt HTTP/1.1 | 29.998261 |

**Figura 6.** *Requisições da etapa de Manutenção de Conectividade*

### Características comuns a ambas as etapas

Um servidor C&C ou o próprio *malware* atualizado podem redirecionar o *bot* para se conectar a outro servidor. Esse comportamento é customizável para os casos de migração de servidor, o que é útil em casos do *botmaster* utilizar estratégias de evasão de conexões (Feily *et al*., 2009). Os métodos para tratamento de comando atendem a esta demanda.

O emulador possui um arquivo de configuração. Nele, há diversas opções de valores configuráveis de variáveis (parâmetros para execução do *bot*) utilizadas em várias partes do emulador. Isso provê maior flexibilidade na caracterização do tráfego a ser gerado.

A requisição do *bot* pode possuir um *payload* que caracteriza o envio de dados roubados do *bot* ao *botmaster*. Como esta também é uma característica comum, está declarada no emulador a classe "*Cipher*" que engloba os algoritmos de criptografia mais utilizados, como AES, DES, RSA, RC4, dentre outros. Tanto o nome

do algoritmo quanto o tamanho da chave, em bits, são lidos do arquivo de configuração do emulador. Para mais de uma camada de criptografia, é necessária a concretização do método que trata da criptografia de *payload*.

A temporização das mensagens trocadas pelo *bot* é feita através da implementação da classe "*Timer*". Durante a execução do *bot*, todas as mensagens indicativas de ações realizadas pelo emulador são gravadas em um arquivo de *log* que é atualizado com precisão na ordem de segundos.

O emulador está implementado na linguagem JAVA, por ser bastante conhecida e por possuir um elevado número de bibliotecas que implementam funcionalidades das mais diversas. A biblioteca "HTTPRequest" foi utilizada na implementação das requisições. Além disso, já está pronto o gerenciamento de marcação de tráfego em nível de aplicação. Isso foi feito, adicionando-se um campo no cabeçalho do protocolo HTTP. Esta marcação pode ser customizada.

Os pontos de extensão do emulador caracterizam o comportamento do *bot*. Alguns destes pontos estão implementados e outros estão somente abstratos. Os que estão concretizados são os que caracterizam mais fundamentalmente o tráfego gerado pelo emulador. Assim, torna-se possível a geração básica de tráfego sem a necessidade de implementações adicionais no emulador. Ou seja, o código com o comportamento fundamental do *bot* nas etapas de atualização e manutenção de conectividade já está implementado e pode ser utilizado para um experimento inicial. Maiores caracterizações de comportamentos do *bot* e suas funcionalidades necessitam ser codificadas. Quaisquer erros de preenchimento dos campos deste arquivo abortam a execução do emulador.

Uma vantagem que é facilmente destacada é o fato do tráfego deste emulador já ser misturado ao tráfego real da máquina que ele está rodando, naturalmente. Isso é exatamente o que ocorre num cenário real com a execução de um *bot*. A mistura do tráfego emulado com o real da máquina se dá porque o emulador proposto roda em um ambiente real.

Destacam-se no emulador proposto: o comportamento do *bot* contido no *trace* emulado, as características mais comuns e abrangentes do *bot* na arquitetura centralizada, o ambiente real de execução, o *log* das atividades, os pontos de extensão na implementação, a isenção de responsabilidade do pesquisador em instanciar um *bot* real e a possibilidade de coleta do tráfego emulado em vários pontos da rede ao mesmo tempo.

## EXPERIMENTO E VALIDAÇÃO

A implementação do emulador visa verificar seu funcionamento correto para, a partir daí, experimentos serem escalados em níveis mais complexos. O experimento serve como ponto de partida para estudos futuros de caracterizações dos *bots* (individualmente) e da *botnet* (o todo). Esse ponto inicial é fundamental para atestar a viabilidade da solução proposta com este emulador.

### Configuração do Cenário

No experimento, o mecanismo *pull* de conexão foi adotado, ou seja, toda ini-

ciativa de conexão partiu do *bot* (cliente) para o C&C (servidor). O servidor atuou passivamente, provendo os arquivos requisitados e respondendo com o devido código nos casos de inexistência de arquivo. Futuramente, mais servidores poderão ser adicionados com seus respectivos endereços (DNS ou IP) devidamente configurados no emulador para o caso de um cenário de maior complexidade no número de conexões e seus redirecionamentos.

Foram configuradas as etapas de comunicação para atualização e manutenção de conectividade. A primeira etapa contou com requisições realizadas para obtenção de arquivo e foram configuradas com o método GET e o envio de mensagem de sucesso após o *download* bem-sucedido (recebimento da mensagem "HTTP 200 OK"). O intervalo entre essas requisições foi configurado para 2 minutos, ou 120 segundos (implementação em segundos). A primeira requisição para atualização foi a primeira a ser executada pelo emulador.

A segunda etapa de manutenção de conectividade possuiu dois tipos de conexão. Uma com a utilização do método GET, também para obtenção de arquivo, e outra com o método POST para emulação do envio de dados do *bot* ao C&C. Durante a pesquisa nos trabalhos anteriormente citados, esses métodos podem coexistir numa *botnet* ou pode ser adotado apenas um dos mesmos para a manutenção de conectividade. O emulador visa abranger as características gerais e, por isso, sua implementação permite que um ou ambos os métodos sejam utilizados para tal.

Refinando mais ainda este comportamento, foi adicionado um temporizador para cada fluxo de requisição por método. Em outras palavras, o intervalo entre as requisições com o método GET possui um valor configurável "x" e o intervalo entre as requisições com o método POST possui um valor configurável "y", que também pode ser igual a "x". Isso permite uma independência entre os fluxos, já que os mesmos podem ser realizados concomitantemente e seus objetivos são distintos.

Vale ressaltar que uma atualização deve ser feita numa temporização maior que a de manutenção de conectividade, já que a disponibilidade de arquivos com novas atualizações são feitas na ordem de dias. A checagem de atualização, portanto, geralmente é feita na ordem de horas ou de dias, dependendo da *botnet*. A conectividade, por sua vez, deve ser checada em uma ordem de tempo inferior, podendo ser segundos ou minutos ou, mais raramente, horas.

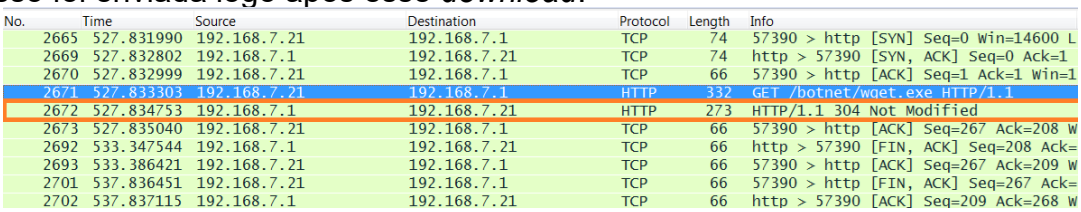
Para ser mantida uma coerência entre os fluxos de requisição para atualização e conectividade, foi feita uma separação desses conjuntos de fluxos por sinalizadores entre os conjuntos. Foi arbitrado, em nível de programação, uma sinalização feita por temporizador de atualização, já que ocorre num espaço de tempo maior que os temporizadores de manutenção de conectividade. Então, quando o temporizador de atualização chega a sua contagem final, são disparados eventos para suspensão das requisições de conectividade até que a atualização (ou o término de sua checagem) seja feita. Ao final, o temporizador de atualização é inicializado novamente e os fluxos de manutenção de conectividade são retomados e seus temporizadores também reinicializados. Em ambos os experimentos, foi adotado que os arquivos buscados pelo *bot* existiriam sempre no servidor. Assim, códigos de exceção de inexistência de arquivo ou outras situações não são esperados.

Toda essa descrição do funcionamento dos fluxos se fez necessária por conta da validação qualitativa dos *traces*. Esta foi realizada através da verificação da existência, do sequenciamento, da repetição e da temporização dos diálogos entre o *bot* e o C&C.

**Condução do Experimento**

O cenário do experimento é simples e foi conduzido num ambiente contendo uma máquina cliente na rede local que representa o *bot* e uma máquina com servidor *web* em rede externa a do *bot*. Os intervalos de tempo das requisições do *bot* foram previamente configurados no arquivo de configuração do emulador.

A etapa de atualização foi configurada para ter uma mensagem de sucesso de *download* bem-sucedido. No caso da não ocorrência de *download*, nenhuma mensagem adicional é enviada. A checagem de nova versão do *malware* para *download* é feita na própria requisição no campo "If-modified-since" do cabeçalho do HTTP. Se houve modificação desde a última data da versão anterior do *malware*, significa que houve a atualização do arquivo com o executável do *malware*, então, o arquivo é baixado e, no caso de *download* bem-sucedido, uma mensagem adicional de sucesso é enviada ao C&C. Depois disso, os fluxos de manutenção de conectividade são iniciados. Se não houve modificação da versão do *malware* desde a última checagem, são iniciados os fluxos de manutenção de conectividade imediatamente. No cenário deste experimento, nenhuma modificação da versão do *malware* ocorreu, conforme o assinalado na resposta "*304 Not Modified*" da Figura 5. Sendo assim, apenas um (o primeiro) *download* foi realizado e uma única mensagem de sucesso foi enviada logo após esse *download*.



| No. | Time | Source | Destination | Protocol | Length | Info |
|---|---|---|---|---|---|---|
| 2665 | 527.831990 | 192.168.7.21 | 192.168.7.1 | TCP | 74 | 57390 > http [SYN] Seq=0 Win=14600 L |
| 2669 | 527.832802 | 192.168.7.1 | 192.168.7.21 | TCP | 74 | http > 57390 [SYN, ACK] Seq=0 Ack=1 |
| 2670 | 527.832999 | 192.168.7.21 | 192.168.7.1 | TCP | 66 | 57390 > http [ACK] Seq=1 Ack=1 Win=1 |
| 2671 | 527.833303 | 192.168.7.21 | 192.168.7.1 | HTTP | 332 | GET /botnet/wget.exe HTTP/1.1 |
| 2672 | 527.834753 | 192.168.7.1 | 192.168.7.21 | HTTP | 273 | HTTP/1.1 304 Not Modified |
| 2673 | 527.835040 | 192.168.7.21 | 192.168.7.1 | TCP | 66 | 57390 > http [ACK] Seq=267 Ack=208 W |
| 2692 | 533.347544 | 192.168.7.1 | 192.168.7.21 | TCP | 66 | http > 57390 [FIN, ACK] Seq=208 Ack= |
| 2693 | 533.386421 | 192.168.7.21 | 192.168.7.1 | TCP | 66 | 57390 > http [ACK] Seq=267 Ack=209 W |
| 2701 | 537.836451 | 192.168.7.21 | 192.168.7.1 | TCP | 66 | 57390 > http [FIN, ACK] Seq=267 Ack= |
| 2702 | 537.837115 | 192.168.7.1 | 192.168.7.21 | TCP | 66 | http > 57390 [ACK] Seq=209 Ack=268 W |

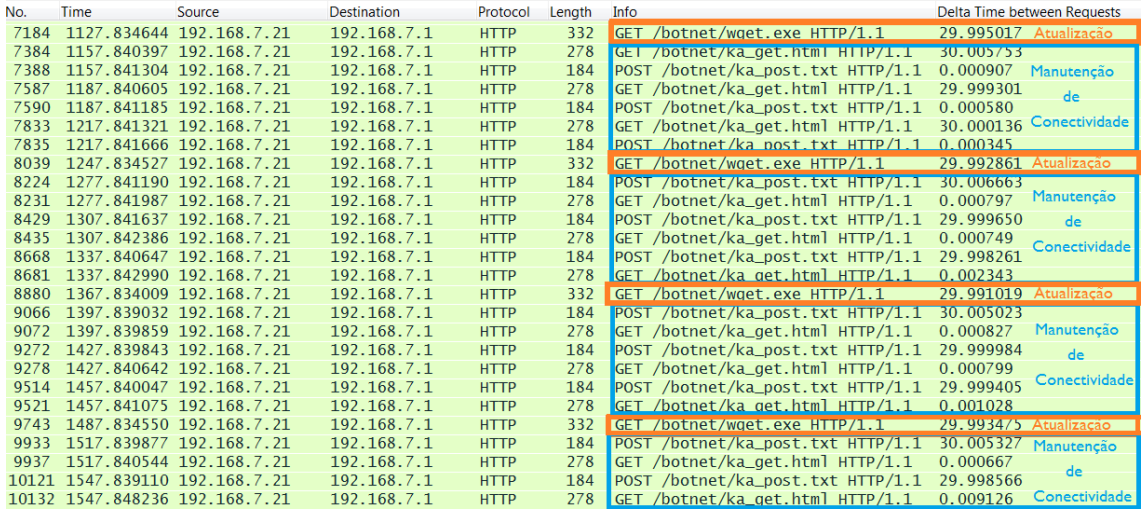
**Figura 7.** *Posteriores tentativas de download do arquivo de atualização do malware*

Vale ressaltar que os campos do cabeçalho HTTP, assim como o *payload* dos pacotes enviados pelo *bot* são configuráveis, sendo, portanto, customizáveis de acordo com o *bot* que se deseja emular. Os campos do cabeçalho HTTP já parametrizados visam proporcionar um melhor uso do emulador com as características mais comuns em *botnets*. Eles podem ser utilizados diretamente em uma emulação básica ou modificados de acordo com o perfil do *bot*.

Dando prosseguimento à descrição dos fluxos, a etapa de manutenção de conectividade é iniciada logo após o término do último fluxo da etapa de atualização. Neste momento, dois tipos de fluxo podem ser configurados, o do método GET e o do método POST. Suas configurações englobam o conteúdo do *payload*, o cabeçalho do HTTP e sua temporização. Neste cenário, ambos os fluxos foram escolhidos para ocorrerem a cada 30 segundos. A temporização inicia-se após o término dos fluxos de atualização. Ou seja, um fluxo de manutenção de conectividade ocorre após 30 segundos do término de todos os fluxos para atualização, num exemplo com esta temporização de 30 segundos. Já a temporização de atualiza-

ção é iniciada imediatamente após o término de seu último fluxo.

O experimento rodou por 10 minutos. O esperado contempla dois fluxos de atualização para a primeira ocorrência de atualização (um fluxo de *download* e outro de mensagem de sucesso) e, após, somente um de atualização na temporização de 120 segundos (requisição sem *download*). Após o término dos fluxos de atualização, os fluxos de GET e POST de manutenção de conectividade foram inciados com 30 segundos de intervalo.



| No. | Time | Source | Destination | Protocol | Length | Info | Delta Time between Requests | |
|---|---|---|---|---|---|---|---|---|
| 7184 | 1127.834644 | 192.168.7.21 | 192.168.7.1 | HTTP | 332 | GET /botnet/wget.exe HTTP/1.1 | 29.995017 | Atualização |
| 7384 | 1157.840397 | 192.168.7.21 | 192.168.7.1 | HTTP | 278 | GET /botnet/ka_get.html HTTP/1.1 | 30.005753 | |
| 7388 | 1157.841304 | 192.168.7.21 | 192.168.7.1 | HTTP | 184 | POST /botnet/ka_post.txt HTTP/1.1 | 0.000907 | Manutenção |
| 7587 | 1187.840605 | 192.168.7.21 | 192.168.7.1 | HTTP | 278 | GET /botnet/ka_get.html HTTP/1.1 | 29.999301 | de |
| 7590 | 1187.841185 | 192.168.7.21 | 192.168.7.1 | HTTP | 184 | POST /botnet/ka_post.txt HTTP/1.1 | 0.000580 | Conectividade |
| 7833 | 1217.841321 | 192.168.7.21 | 192.168.7.1 | HTTP | 278 | GET /botnet/ka_get.html HTTP/1.1 | 30.000136 | |
| 7835 | 1217.841666 | 192.168.7.21 | 192.168.7.1 | HTTP | 184 | POST /botnet/ka_post.txt HTTP/1.1 | 0.000345 | |
| 8039 | 1247.834527 | 192.168.7.21 | 192.168.7.1 | HTTP | 332 | GET /botnet/wget.exe HTTP/1.1 | 29.992861 | Atualização |
| 8224 | 1277.841190 | 192.168.7.21 | 192.168.7.1 | HTTP | 184 | POST /botnet/ka_post.txt HTTP/1.1 | 30.006663 | |
| 8231 | 1277.841987 | 192.168.7.21 | 192.168.7.1 | HTTP | 278 | GET /botnet/ka_get.html HTTP/1.1 | 0.000797 | Manutenção |
| 8429 | 1307.841637 | 192.168.7.21 | 192.168.7.1 | HTTP | 184 | POST /botnet/ka_post.txt HTTP/1.1 | 29.999650 | de |
| 8435 | 1307.842386 | 192.168.7.21 | 192.168.7.1 | HTTP | 278 | GET /botnet/ka_get.html HTTP/1.1 | 0.000749 | Conectividade |
| 8668 | 1337.840647 | 192.168.7.21 | 192.168.7.1 | HTTP | 184 | POST /botnet/ka_post.txt HTTP/1.1 | 29.998261 | |
| 8681 | 1337.842990 | 192.168.7.21 | 192.168.7.1 | HTTP | 278 | GET /botnet/ka_get.html HTTP/1.1 | 0.002343 | |
| 8880 | 1367.834009 | 192.168.7.21 | 192.168.7.1 | HTTP | 332 | GET /botnet/wget.exe HTTP/1.1 | 29.991019 | Atualização |
| 9066 | 1397.839032 | 192.168.7.21 | 192.168.7.1 | HTTP | 184 | POST /botnet/ka_post.txt HTTP/1.1 | 30.005023 | |
| 9072 | 1397.839859 | 192.168.7.21 | 192.168.7.1 | HTTP | 278 | GET /botnet/ka_get.html HTTP/1.1 | 0.000827 | Manutenção |
| 9272 | 1427.839843 | 192.168.7.21 | 192.168.7.1 | HTTP | 184 | POST /botnet/ka_post.txt HTTP/1.1 | 29.999984 | de |
| 9278 | 1427.840642 | 192.168.7.21 | 192.168.7.1 | HTTP | 278 | GET /botnet/ka_get.html HTTP/1.1 | 0.000799 | Conectividade |
| 9514 | 1457.840047 | 192.168.7.21 | 192.168.7.1 | HTTP | 184 | POST /botnet/ka_post.txt HTTP/1.1 | 29.999405 | |
| 9521 | 1457.841075 | 192.168.7.21 | 192.168.7.1 | HTTP | 278 | GET /botnet/ka_get.html HTTP/1.1 | 0.001028 | |
| 9743 | 1487.834550 | 192.168.7.21 | 192.168.7.1 | HTTP | 332 | GET /botnet/wget.exe HTTP/1.1 | 29.993475 | Atualização |
| 9933 | 1517.839877 | 192.168.7.21 | 192.168.7.1 | HTTP | 184 | POST /botnet/ka_post.txt HTTP/1.1 | 30.005327 | Manutenção |
| 9937 | 1517.840544 | 192.168.7.21 | 192.168.7.1 | HTTP | 278 | GET /botnet/ka_get.html HTTP/1.1 | 0.000667 | de |
| 10121 | 1547.839110 | 192.168.7.21 | 192.168.7.1 | HTTP | 184 | POST /botnet/ka_post.txt HTTP/1.1 | 29.998566 | Conectividade |
| 10132 | 1547.848236 | 192.168.7.21 | 192.168.7.1 | HTTP | 278 | GET /botnet/ka_get.html HTTP/1.1 | 0.009126 | |

**Figura 8.** *Requisições das etapas de atualização do malware e manutenção de conectividade do bot*

### Validação do Experimento

Foram validados qualitativamente os seguintes quesitos dos fluxos: ocorrência, sequenciamento, repetição, temporização e interação das etapas. O resultado da validação está mostrado na Tabela 1. Foi utilizada a ferramenta Wireshark (Wireshark Foundation, 2013) para captura de tráfego de rede (*dump*) com formato/extensão “.pcap”.

A mudança de fluxos para atualização e conectividade foi feita sem interferência, ou seja, não houve fluxos concorrentes de atualização e conectividade. Os fluxos para atualização foram os primeiros a serem executados. Quando ocorreram, os fluxos de conectividade ficaram suspensos por até 60,9s para o método GET e por 60,8s para o método POST. A justificativa dessa suspensão por, aproximadamente, 1 minuto, é devido a interseção dos tempos de intervalo de atualização e conectividade. A atualização ocorre a cada 120s, enquanto que a manutenção de conectividade é a cada 30s. Pode ser percebido facilmente que um tempo é múltiplo do outro. Por isso, a cada 3 iterações dos fluxos de conectividade, o seu 4º fluxo não é executado porque o temporizador de atualização é disparado no segundo 120, aproximadamente. O 4º fluxo de conectividade, neste caso, ocorreria em um espaço de tempo pequeno após o 120º segundo. Por isso, o total de iterações para cada método de manutenção de conectividade não é maior que 15, em 600 segundos de experimento, contados a partir da primeira requisição feita.

Os tempos representados na Tabela 1 possuem um pequeno intervalo de variação menor que 2 segundos. Isso é devido ao fato do experimento rodar em um ambiente real. O tráfego normal de uma máquina conectada a uma rede esteve presente em toda a duração do experimento. Atrasos de conexão são normais quando se tem uma máquina que roda várias aplicações de rede ao mesmo tempo.

No caso deste experimento, a máquina cliente com o *bot* rodou concomitantemente outras aplicações que fizeram conexão à rede (local e externa). Isso justifica o atraso de até 2s no geral das temporizações.

A análise dos fluxos deste experimento desconsiderou quaisquer tráfegos diferentes dos pertinentes a este experimento. Isso se fez necessário para a validação da temporização dos fluxos, principalmente.

**Tabela 1**. *Validação qualitativa dos fluxos do experimento*

| Quesitos do fluxo | Atualização | | Menutenção de Conectividade | |
|---|---|---|---|---|
| | Requisição para *download* | Envio de mensagem de sucesso | Requisição com método GET | Requisição com método POST |
| **Ocorrência** | Presente. | Uma única vez na primeira requisição para *download* do *malware* . | Após 30,1s a 30,3s do término do último fluxo de atualização. | Após 30,1s a 30,2s do término do último fluxo de atualização. |
| **Sequenciamento** | Primeiro fluxo simulado. Posteriormente, ocorreu após os fluxos para manutenção de conectividade. | Após o *download* bem-sucedido da primeira requisição para *download* . | Ocorrência após os fluxos para atualização. Suspensão dos fluxos de conectividade para realização de requisições de atualização e retomada após o seu término. | |
| **Repetição & Temporização** | Ocorreu em intervalos de 119,1s a 120,7s.<br><br>Total de iterações: 5 | Não se aplica, pois é um fluxo dependente. | Intervalos de 29,8s a 30,3s.<br><br>Total de iterações: 15 | Intervalos de 29,9s a 30,2s.<br><br>Total de iterações: 15 |

Todos os quesitos foram verificados e a validação qualitativa foi bem-sucedida. Pode-se concluir, portanto, que o experimento atendeu o esperado. O emulador proposto atende o que está descrito na literatura de trabalhos abrangentes da área de Segurança da Informação, mais especificamente de Defesa Cibernética. Os *traces* de *bot* emulados respeitaram todas as características configuradas dos *bots*. O lado comportamental do *bot* na rede funcionou a contento.

## TRABALHOS RELACIONADOS

A seguir, serão citadas duas ferramentas que geram *traces* de *bot*. As limitações técnicas de ambas e a carência de *traces* com acesso público para fins de estudo serviram de motivador para a construção do emulador proposto neste artigo.

A primeira ferramenta foi produto de um estudo de uma tese de doutorado em 2009, cujo autor é Christopher P. Lee. Sua ferramenta para geração de *trace* de *bot* é um *framework* para emulação e análise de *botnet* ou, como o próprio autor descreve, "*Framework for botnet emulation and analysis*", o Rubot (LEE, 2009). Um dos problemas, ou limitações, é o fato da sua construção ter sido embasada em funcionalidades provenientes das *botnets* Storm e Nugache, ambas P2P. Fazendo-se uma análise primária, pode-se observar que para a construção de métodos mais genéricos que abranjam, de fato, outras *botnets* da arquitetura centralizada, seriam necessários outros modelos de *botnets*, de preferência, os mais representativos de ambas as arquiteturas. Conclui-se que o Rubot não apresenta uma implementação genérica em termos de características de arquitetura de *botnet*, já que foi baseado em dois exemplares de uma única arquitetura. Além disso, o próprio autor relata algumas limitações de seu *framework* e funcionalidades a serem desenvol-

vidas, como o suporte a testes, integrações, modelos de mensagem instantânea e de sensores.

A segunda ferramenta, e também *framework*, para os mesmos fins é o SLINGbot (Jackson *et al.*, 2009). Ele foi construído baseado na taxonomia de um relatório técnico (Trendmicro, 2006). O SLINGbot possui um ambiente totalmente controlado de emulação de *botnet* e, em seus exemplos de *botnets* implementadas, há bastante foco na utilização do mecanismo *push*. Este mecanismo é mais característico de *botnets* da arquitetura descentralizada.

O acesso ao *framework* SLINGbot não é público, ao contrário do Rubot para fins de estudo. Como o motivador do desenvolvimento do emulador proposto neste artigo é a carência de *traces* disponíveis para fins de estudo na área de tráfego malicioso com o foco em atividades de rede, o desenvolvimento do emulador foi baseado não em duas *botnets* (como foi o caso do Rubot) ou em funcionalidades (como o caso do SLINGbot), mas nas características gerais de arquitetura de *botnets*. Isso abrange um maior número de *botnets*. O emulador proposto não é limitado apenas a funcionalidades no *trace* emulado. O seu diferencial é o comportamento do *bot* no *trace* emulado. O comportamento do *bot* está ligado diretamente à sequência das funcionalidades, a temporização, como as etapas são intercaladas e quando elas ocorrem. A elaboração do comportamental é muito mais do que um simples conjunto de funcionalidades que podem ser utilizadas sem critério algum. Os critérios norteiam o comportamental básico do emulador de *bot* na arquitetura centralizada.

## CONSIDERAÇÕES FINAIS

O tráfego emulado foi devidamente validado e os fluxos de atualização e manutenção de conectividade do *bot* funcionaram a contento, respeitando todas as configurações previamente realizadas no arquivo de configuração e nos pontos extensíveis do emulador.

Todos os resultados foram satisfatórios. Para facilitar a geração de tráfego deste emulador, todos os códigos utilizados no experimento estão disponíveis no próprio emulador. Com isso, a base para emulação de *traces* de *bot* está disponível e abre espaço para uma gama de trabalhos futuros.

As principais contribuições deste emulador são detacadas a seguir:

- comportamento do *bot* contido no *trace* emulado, baseado nas características mais comuns e abrangentes do *bot* na arquitetura centralizada;
- customização do *trace* gerado a partir de extensão de código, configuração e parametrização dos arquivos de configuração do emulador.
- emulação de *traces* de *bot* contínua e feita em ambiente real com a mistura do tráfego sintético com o tráfego de aplicações de rede sendo feita naturalmente, assim como um *bot* no mundo real;
- qualidade do *trace* emulado validada;
- possibilidade de coleta de *traces* de *bot* em vários pontos da rede simultaneamente.

O pesquisador que fizer uso dessa ferramenta está insento da responsabilidade de instanciação de um *bot* real. Ele pode controlar o *bot* de acordo com a con-

figuração e a parametrização relativamente simples do emulador. O *trace* emulado com qualidade comprovada, portanto, serve de insumo para seus estudos.

Uma gama de trabalhos pode ser desenvolvida a partir deste emulador. A arquitetura descentralizada, ou distribuída (P2P), poderá ser estudada futuramente para também ser instanciada neste emulador. Destacam-se os pontos de extensão do emulador para futuras implementações de novas funcionalidades e caracterização do *bot*. Afinal, as características do *bot* são o que diferenciam um conjunto de *bots* de outro, ou seja, uma *botnet* da outra.

Técnicas de autenticação de conexão e de criptografia da URL na requisição do emulador, mecanismo *push* de comunicação e melhorias de marcação de tráfego em outras camadas no protocolo HTTP ficam em aberto e poderão ser desenvolvidas futuramente. Da mesma forma, a adição de marcação de tráfego em outros protocolos a serem adicionados no emulador proposto.

As atividades de ataque e propagação podem ser adicionadas em paralelo ou de forma sequencial nos fluxos de execução do emulador de *traces* de *bot*. Isso enriquece o *trace* a ser gerado e abrange mais características do *bot* em várias etapas de seu ciclo de vida.

Uma validação quantitativa poderá ser feita futuramente com a comparação do *trace* emulado com *traces* reais de *bots*. Isso enriquecerá a qualidade do emulador e do *trace* a ser gerado. Adicionalmente, poderão ser realizados testes para fins de detecção do *trace* emulado. A princípio, os métodos de detecção existentes estão mais voltados para as atividades de ataque numa rede, o que é uma etapa aberta no presente trabalho. Uma maior sofisticação destes métodos, de acordo com as etapas configuradas no emulador, poderá acionar a detecção de um *bot*. Tais sofisticações estarão relacionadas com o comportamento do *bot* em seus ajustes mais finos de temporização e diálogos do *bot*, além de outras extensões que poderão ser feitas.

O emulador proposto é de acesso público para fins de estudo acadêmico na área de Segurança da Informação, subárea Defesa Cibernética. A solicitação do emulador deve ser feita aos autores do presente artigo. Os *traces* produzidos pelo mesmo podem e devem ser públicos para fins de estudo. O compartilhamento do conhecimento, assim, é realizado racionalmente.

## REFERÊNCIAS BIBLIOGRÁFICAS


- *ABT, S. Missing Data Problem in Cyber Security Research. Tese de Doutorado, Biometrics and Internet Security Research Group Hochschule Darmstadt, Darmstadt, Alemanha,* ***2013****.*
- *CERON, J. M.; Granville, L. Z.; Tarouco, L. M. R. Uma arquitetura baseada em assinaturas para mitigação de botnets. Simpósio Brasileiro em Segurança da Informação e de Sistemas Computacionais (SBSeg),* ***2010****.*
- *CHOI, H.; Lee, H.; Kim, H. BotGAD: detecting botnets by capturing group activities in network traffic. International ICST Conference on COMmunication System softWAre and middlewaRE (COMSWARE), ACM, Nova York, Estados Unidos,* ***2009****.*
- *COOKE, E.; Jahaniana, F.; Mcpherson, D. The zombie roundup: understanding, detecting, and*

*disrupting botnets. Steps to Reducing Unwanted Traffic on the Internet Workshop, USENIX Association, Berkeley, CA, Estados Unidos,* ***2005****.*

- *FEILY, M.; Shahrestani, A.; Ramadass, R. A Survey of Botnet and Botnet Detection. International Conference on Emerging Security Information, Systems and Technologies,* ***2009****.*
- *GONCHAROV, M. Russian Underground 101, 2012. Disponível em: <http://www.trendmicro.com/cloud-content/us/pdfs/securityintelligence/white-papers/wp-russian-underground-101.pdf>. Trend Micro Incorporated Research Paper. Acessado em: 27/09/****2013****.*
- *GU, G.; Zhang, J.; Lee, W. BotSniffer: Detecting Botnet Command and Control Channels in Network Traffic. Annual Network and Distributed System Security Symposium, The Internet Society,* ***2008****.*
- *LEE, C. P.; Framework for botnet emulation and analysis; Tese de Doutorado, School of Electrical and Computer Engineering in Georgia Institute of Technology, Estados Unidos,* ***2009****.*
- *JACKSON, A. W., Lapsley, D., Jones, C., Zatko, M., Golubitsky, C.; Strayer, W. T.; SLINGbot: A System for Live Investigation of Next Generation Botnets. Proceedings of Cybersecurity Applications and Technologies Conference for Homeland Security (CATCH), Washington, Estados Unidos,* ***2009****.*
- *SADHAN, B. A.; Moura, J. M. F.; Lapsley, D., Jones, C.; Strayer, W. T. Detecting botnets using command and control traffic. IEEE International Symposium on Network Computing and Applications,* ***2009****.*
- *SILVA, S. S., Silva, R. M., Pinto, R. C.; Salles, R. M. Botnets: A survey. Computer Networks,* ***2012****.*
- *STONE-GROSS, B., Cova, M., Cavallaro, L., Gilbert, B., Szydlowski, M., Kemmerer, R., Kruegel, C.; Vigna, G. Your botnet is my botnet: analysis of a botnet takeover. ACM conference on Computer and communications security, Nova York, NY, Estados Unidos,* ***2009****.*
- *STURGEON, W. Net pioneer predicts overwhelming botnet surge. Technical report, CNET News.com, 2007. Disponível em: <http://news.cnet.com/2100-7348_3-6154221.html>. Acessado em: 23/09/****2013****.*
- *TRENDMICRO. Taxonomy of Botnet Threats. Technical report, November 2006. Disponível em: <http://www.cs.ucsb.edu/kemm/courses/cs595G/TM06.pdf>. Acessado em: 21/10/****2013****.*
- *Tyagi, A. K.; Aghila, G. A Wide Scale Survey on Botnet. International Journal of Computer Applications,* ***2011****.*
- *WANGA, P.; Aslam, B.; Zou, C. C. Peer-to-Peer Botnets. Editora Springer,* ***2010.***
- *WIRESHARK - Go Deep. Wireshark Foundation,* ***2014****. Disponível em: https://www.wireshark.org/ Acessado em 10/10/****2013****.*
- *YONG, W., Tefera, S. H., Beshah, Y. K.; Understanding Botnet: From Mathematical Modelling to Integrated Detection and Mitigation Framework. ACIS International Conference on Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing,* ***2012****.*
- *ZEIDANLOO, H. R.; Manaf, A. A.; Botnet Command and Control Mechanisms; Second International Conference on Computer and Electrical Engineering, IEEE Computer Society. Washington, Estados Unidos,* ***2009****.*
- *ZHU, Z., Lu, G., Chen, Y., Fu, Z. J., Roberts, P.; Han, K.; Botnet research survey. Computer Software and Applications, 32nd Annual IEEE International,* ***2008****.*
- *ZOU, C. C.; cunningham, R.; Honeypot-aware advanced botnet construction and maintenance. International Conference on Dependable Systems and Networks,* ***2006****.*