Monitoração de comportamento de *malware* em sistemas operacionais Windows NT 6.x de 64 bits

Marcus Botacin[1,3], Vitor Afonso[1], Paulo Lício de Geus[1], André Grégio[1,2]

[1] Instituto de Computação (IC)
Universidade Estadual de Campinas (Unicamp)
Campinas – SP – Brasil

[2]Divisão de Segurança de Sistemas de Informação (DSSI)
Centro de Tecnologia da Informação Renato Archer (CTI)
Campinas – SP – Brasil

[3]Bolsista PIBIC-CNPq

{marcus, vitor, paulo}@lasca.ic.unicamp.br, andre.gregio@cti.gov.br

Abstract. *Malware are persistent threats to systems security that are constantly evolving to prevent detection and dynamic analysis techniques. Currently, there is no known dynamic analysis system (publicly available or described in the literature) that supports 64-bits malware (PE+ format). It is difficult to monitor malware for Windows NT 6.x due to new security mechanisms introduced in these systems, making it expensive to build or port an actual analysis system/tool. In this paper, we present the design and implementation of a novel malware dynamic analysis system for Windows 8, as well as the obstacles and challenges we faced. We present the tests and results of the proposed system, evaluated with 2,937 32 and 64-bit malware samples.*

Resumo. *Programas maliciosos (*malware*) são ameaças persistentes à segurança, evoluindo constantemente para evitar a detecção e análise dinâmica. Atualmente, nenhum dos sistemas descritos na literatura ou disponíveis publicamente suportam* malware *de 64 bits (PE+). A monitoração de* malware *em Windows NT 6.x é dificultada devido à introdução de novos mecanismos de segurança, tornando custosa a construção ou portabilidade de um sistema de análise funcional. Neste artigo, apresenta-se o projeto e a implementação de um sistema de análise dinâmica de* malware *em Windows 8, os obstáculos e desafios encontrados e sua avaliação com 2.937 exemplares de 32 e 64 bits.*

1. Introdução

Programas maliciosos têm sido a ameaça mais grave e persistente para a segurança de sistemas interconectados em rede e seus usuários. Esses programas, genericamente chamados de *malware*, subvertem a operação legítima de um sistema computacional de modo a violar sua integridade, confidencialidade ou disponibilidade. Ataques por *malware* são motivados por inúmeros resultados espúrios, tais como evasão de informações, roubo de credenciais, forja de identidade, armazenamento de conteúdo ilícito, lançamento de ataques contra terceiros e ganhos financeiros. Durante um ataque, traços de atividades feitas pelo exemplar de *malware* podem ser observados no sistema operacional. Como exemplo desses traços, pode-se citar arquivos obtidos da Internet, substituição de bibliotecas

ou aplicações do sistema, modificação de chaves do Registro do sistema operacional e alteração em configurações relacionadas com mecanismos de segurança e atualização. A captura desses traços envolve a interceptação das ações realizadas por um programa monitorado no sistema operacional alvo e depende de vários fatores, incluindo o nível no qual a interceptação irá ocorrer (do usuário, do *kernel* ou ainda na camada entre o sistema de virtualização e o sistema base), os mecanismos de proteção existentes no próprio sistema operacional e a capacidade do *malware* analisado em evadir a monitoração.

Os sistemas operacionais Windows ainda são os alvos principais dos criadores de *malware* para computadores pessoais. Embora os sistemas Windows NT com versões acima de 6 (Vista, 7, 8 e 8.1) introduzam diversos mecanismos novos para aumentar a segurança, eles apresentam compatibilidade com aplicações feitas para NT 5, podendo portanto executar programas em 32 e 64 bits. Devido às diferenças entre os sistemas operacionais Windows NT 5.1 (XP) e NT 6.2 (8) ocorrerem em vários níveis (*kernel*, mecanismos de segurança, modo de execução), faz-se necessária a compreensão de como os exemplares de *malware* se comportam durante a execução no Windows 8 e como se dá a interação entre o sistema operacional, o programa malicioso e a ferramenta de monitoração. Mais do que isso, é necessário projetar e implementar uma nova ferramenta de monitoração de comportamento de execução de programas para atuar em sistemas de 64 bits, dado que os analisadores de *malware* publicamente disponíveis não estão prontos para monitorar programas maliciosos de 64 bits.

Neste artigo propõe-se um sistema de análise dinâmica de *malware* baseado em Windows 8. As principais contribuições deste trabalho são: o levantamento e a descrição dos novos mecanismos de segurança presentes em Windows NT 6.x e como estes afetam o desenvolvimento de uma ferramenta para monitoração de ações de programas em execução; o projeto e a implementação de uma arquitetura de análise de *malware* para sistemas e exemplares de 64 bits, com as decisões tomadas, desafios e especificações; a identificação do comportamento suspeito observado nos resultados de execução de mais de 2 mil exemplares de *malware* obtidos nos testes para validação do sistema proposto, bem como a constatação de que *malware* codificado (e compilado) para sistemas Windows XP de 32 bits é capaz de infectar os sistemas novos de 64 bits.

O restante do artigo é dividido da seguinte forma: a Seção 2 trata dos aspectos técnicos distintos introduzidos pela versão NT 6.x dos sistemas operacionais Windows e suas implicações na implantação de uma ferramenta de monitoração de chamadas de sistema e de API nos níveis de *kernel* e de usuário; na Seção 3, são discutidos os principais trabalhos relacionados com sistemas de análise dinâmica de *malware* voltados para obtenção do comportamento, as técnicas utilizadas em sua implementação e as limitações encontradas; na Seção 4, os detalhes do projeto e implementação da arquitetura proposta são expostos, bem como as decisões tomadas para balancear as vantagens e desvantagens das técnicas de monitoração, abrangência de captura e flexibilidade de aplicação (emulação e *bare-metal*); a Seção 5 apresenta os testes realizados na validação do funcionamento do sistema, em especial da ferramenta desenvolvida para monitoração das ações em nível de *kernel*, além de mostrar os resultados da análise de mais de dois mil exemplares coletados recentemente e em atividade. A conclusão do artigo está na Seção 6.

2. Aspectos Técnicos do NT 6.x

A família de sistemas operacionais Windows versão NT 6.x—iniciada com o Windows Vista e da qual o Windows 8 faz parte—traz uma série de diferenças em relação ao Windows XP. A compreensão desses novos mecanismos de segurança e modos de operação é fundamental para se projetar uma ferramenta de monitoração de programas, pois eles afetam diretamente a operação de aplicações no nível do usuário e do *kernel*. Nessa seção, são discutidas as diferenças introduzidas nos Windows modernos e suas implicações para o desenvolvimento de um sistema de análise de *malware*.

2.1. Kernel Patch Protection (KPP)

A proteção de *patch de kernel*, como definida em [Microsoft 2013c], proíbe que *drivers* em nível privilegiado estendam ou substituam serviços do *kernel* por meios não documentados. Tal proibição visa aumentar a segurança do sistema, uma vez que, além dos usos por programas legítimos, muitos *rootkits* utilizam-se desta possibilidade para infectar o sistema. Este mecanismo de proteção de *kernel* encontra-se presente apenas nas versões 64 bits do sistema operacional, dado que para as versões de 32 bits existe uma gama enorme de aplicativos lançados baseados em tais *patches* que viriam a se tornar incompatíveis caso o mecanismo fosse incorporado a todas as edições do sistema. Deve-se notar que a base instalada de aplicativos dependentes destes *patches* para a arquitetura de 64 bits é significativamente menor. O referido mecanismo visa impedir:

- Modificações nas tabelas de serviços do sistema, por exemplo, conectar-se à tabela `KeServiceDescriptor`;
- Modificações na IDT (*Interrupt Descriptor Table*);
- Modificações na GDT (*Global Descriptor Table*);
- Uso de pilhas de *kernel* que não sejam alocadas por este;
- Aplicar *patches* em qualquer parte do *kernel* (somente AMD64).

Uma implicação que KPP traz para a monitoração dinâmica de *malware* é na implementação de técnicas de *hooking*, as quais consistem na alteração das funções originais do sistema por funções de interceptação, responsáveis por coletar os dados monitorados e dar continuidade às ações originalmente pretendidas. Essa técnica, quando aplicada no nível do *kernel*, se mostra altamente efetiva, uma vez que captura dados em um nível de execução privilegiado, está devidamente isolada do espaço de execução do *malware* e afeta todo o sistema, não necessitando ser aplicada a cada processo individualmente.

A implementação de *hooks* de *kernel* é realizada através da substituição dos endereços das funções diretamente nas tabelas exportadas pelo *kernel*. Tal substituição é dificultada no Windows 8 devido ao mecanismo de KPP, dado que quando se tenta implementar esse tipo de *hooking* em 64 bits, verifica-se que as tabelas de funções não são mais exportadas pelo *kernel*. Além disso, sua localização é imprevisível, uma vez que há um mecanismo de aleatorização de espaço de memória. Desta forma, deve-se identificar novos meios de interceptação no nível do *kernel* que não sejam baseados em *hooking*.

Uma outra forma de interceptar APIs do Windows é através do uso de *Detours*, uma biblioteca provida pela Microsoft. Sua utilização permite realizar modificações dinâmicas (durante a execução do programa) no início da função que se deseja interceptar através da inserção de uma instrução *assembly* de pulo incondicional (`JMP`). O uso de *Detours*, no entanto, passou a ser condicionado a aquisição de licenças [Microsoft 2013b].

2.2. Assinatura de *Drivers*

Com o objetivo de aumentar a segurança no sistema de modo a evitar que componentes arbitrários sejam carregados no *kernel*, a Microsoft passou a exigir que os *drivers* sejam assinados digitalmente. Tal política de desenvolvimento impede, a princípio, a utilização de um *driver* como mecanismo de captura se este não for assinado digitalmente, o que não é compatível com os requisitos de uma ferramenta de monitoração. No entanto, esta proteção pode ser desligada no ambiente de análise, pois os exemplares de *malware* em geral atuam em espaço de usuário e não apresentam a característica de carregar *drivers* no sistema. Entretanto, o desligamento desse sistema facilita a atuação de *rootkits*, sendo que estes, assim como ocorre em todos os sistemas tradicionais de análise dinâmica de *malware*, não tem sua execução monitorada pela ferramenta proposta.

2.3. Sessões

De modo a tentar isolar diferentes famílias de aplicações (aplicações gráficas, serviços de sistema, serviços remotos), o sistema operacional Windows passou a implementar o conceito de sessões, no qual os dados e privilégios de execução de cada uma dessas famílias ficam restritos às mesmas. Essa mudança trouxe impactos significativos, como o impedimento do lançamento de *threads* remotas, comumente usadas para injeções de *Dynamic Link Library* (DLL) [Microsoft 2013a]. Dessa forma, minimiza-se a chance de sucesso dos ataques por injeção de DLL aos *browsers*, ao mesmo tempo em que se dificulta o monitoramento de atividades suspeitas através de *DLL hooking*.

2.4. Mudanças na API

As diferenças arquiteturais promovidas do Windows XP para o Windows Vista, e consequentemente para o Windows 8, trouxeram mudanças nas interfaces de programação. Embora as interfaces antigas, em geral, ainda funcionem, deve-se realizar a migração para as novas interfaces de forma que se possa utilizar todos os recursos disponíveis. Uma mudança significativa da API pode ser vista em [Microsoft 2014a] e [Microsoft 2014b].

As interfaces providas pelo próprio sistema operacional provêem a possibilidade de uso na interceptação das ações realizadas e, consequentemente, na análise dinâmica de *malware*. Ao se projetar uma ferramenta de análise, deve-se atentar para as interfaces utilizadas pois estas podem sofrer modificações nas atualizações do sistema, dificultando a portabilidade do mecanismo de monitoração.

Interfaces como *callbacks* e *filters* estão disponíveis a partir do *kernel* por meio do uso de *drivers*. Por serem providos pelo sistema operacional, *callbacks* e *filters* apresentam interfaces e estruturas bem definidas, além de serem bem documentados, o que facilita o desenvolvimento. Adicionalmente, não requerem qualquer mudança em estruturas internas do *kernel* ou de bibliotecas dinâmicas, possuindo suporte nativo no sistema. A desvantagem no uso dessas técnicas é que a captura de informações é limitada às funcionalidades providas pelas interfaces utilizadas.

3. Trabalhos Relacionados

Há diversas ferramentas para a monitoração do comportamento de execução de programas maliciosos, as quais aplicam diferentes técnicas para interceptar as ações efetuadas. Nesta

seção, serão avaliados alguns sistemas de análise dinâmica de *malware* e suas características principais, ressaltando que todos eles suportam apenas executáveis de 32 bits.

Anubis [Bayer et al. 2006] é um sistema de análise dinâmica que se utiliza da técnica de *Virtual Machine Introspection* (VMI) aplicada ao emulador Qemu [Bellard 2005]. Com essa técnica, cria-se uma camada entre o sistema de análise (*guest*) e o ambiente-base (*host*) para processamento e controle, possibilitando que a interceptação das ações executadas pelo *malware* dentro do ambiente de análise seja feita sem que haja qualquer interferência neste. Isso torna possível que um dado *malware* seja analisado sem qualquer tipo de modificação "visível", seja no sistema *guest*, seja no próprio *malware*. O sistema operacional utilizado pelo Anubis no ambiente de análise é o Windows XP SP3, do qual são capturados diversos tipos de atividade durante a execução do *malware* (sistema de arquivos, processos, Registro, objetos de sincronização e tráfego de rede). Ao final da execução, é produzido um relatório técnico sobre a análise do exemplar submetido.

No CWSandbox [Willems et al. 2007], a captura das informações é realizada por uma DLL, a qual precisa ser injetada no processo do *malware*. Quando a DLL é carregada, as principais funções utilizadas para fazer a interface entre o programa e o sistema de análise (por exemplo, modificações em arquivos) têm seu início modificado. Desta forma, um desvio incondicional é executado assim que uma dada função é chamada. Para iniciar o procedimento de análise, existe um componente dentro do ambiente de monitoração—`c`wsandbox.exe—cujas funções são criar o processo do *malware* em estado suspenso, injetar a DLL e retomar a execução do processo em questão. Além disso, este componente é informado caso o *malware* inicialize ou modifique algum processo, para que a DLL de monitoração seja injetada neles também. Finalizada a análise, é gerado um relatório em diversos formatos (HTML, XML e texto), o qual contém as ações realizadas pelo *malware* no sistema operacional monitorado.

Cuckoo [Guarnieri 2013] utiliza uma técnica conhecida como *inline hooking* para interceptar as chamadas de sistema executadas pelo programa a ser monitorado. Para implementar *inline hooking*, Cuckoo precisa carregar uma DLL no processo que se deseja monitorar. O *hooking* é implementado de forma específica para cada função interceptada, diferentemente de um simples salto incondicional no início da função. Isto dificulta métodos triviais de detecção, buscando evitar assim que a análise não seja bem sucedida. Atualmente, existem dois métodos de monitoração implementados, os quais são escolhidos de forma aleatória no momento que o *inline hooking* é instalado. A DLL do Cuckoo é carregada por um *script* em linguagem Python que permanece em execução no ambiente de análise durante todo o processo. Além da tarefa de carregamento, tal *script* é notificado caso o exemplar de *malware* monitorado modifique ou crie um novo processo durante a análise, indicando que a DLL de monitoração deve ser carregada nele. O Cuckoo tem seu código disponível para utilização (`http://www.cuckoosandbox.org`), a qual requer a preparação do ambiente com uma instalação local para a realização das análises.

BehEMOT [Filho et al. 2010] é uma ferramenta de análise dinâmica de *malware* para Windows XP que pode monitorar as principais interações entre um exemplar de *malware* (e seus processos-filhos) e o sistema operacional alvo, como operações em arquivos, chaves do Registro, processos e objetos de sincronização (*mutex*). Além da análise comportamental do *malware* no sistema operacional, a ferramenta realiza a captura de tráfego de rede, provendo informações sobre a interação do exemplar monitorado com

o ambiente externo, como servidores comprometidos hospedando outros objetos maliciosos ou armazenando informações sensíveis. Visando contornar as possíveis técnicas de anti-análise utilizadas por alguns *malware* para detectar máquinas virtuais, a ferramenta baseia-se em uma arquitetura mista de ambientes emulado e real, de modo que os exemplares que não podem ser analisados no ambiente emulado são encaminhados para o ambiente em *bare metal*. Esta característica de flexibilidade entre os ambientes é alcançada devido ao componente de monitoração ter sido implementado como um *driver* de *kernel* que aplica a técnica de *SSDT hooking*, que permite capturar uma vasta gama de ações em nível privilegiado por meio de chamadas de sistema nativas.

Capture-BAT [Seifert et al. 2007] é uma ferramenta de análise de *malware* baseada em *drivers* de *kernel*, objetivando, sobretudo, a portabilidade. O Capture-BAT executa em Windows XP SP2 e monitora as operações de "READ" e "WRITE" através de um *filesystem filter*, e a criação/término de processos e as operações em chaves de Registro através de *kernel callbacks*. Deve-se destacar que, diferentemente das ferramentas descritas anteriormente, não há propagação das ações sobre processos em relação aos demais componentes, isto é, a captura das informações é feita em modo "*system-wide*". Além disso, tanto no caso dos processos, quanto no do Registro, armazena-se apenas algumas informações dentre as possíveis de serem coletadas, como o *timestamp* (em ambos os casos), o *path* do processo (em sua *callback*) e o *path* da chave (no monitor de registro). Portanto, informações mais detalhadas, como o valor de uma chave escrita no Registro e os parâmetros passados para um dado processo, ficam a cargo de outros programas.

3.1. Considerações sobre as ferramentas avaliadas

Com base nos trabalhos relacionados e suas características de operação, as seguintes considerações foram elencadas:

- Anubis fornece um bom referencial para obter abrangência de monitoração e exibição dos resultados. No entanto, verifica-se que apesar do sistema operacional "*guest*" não sofrer alterações, o mecanismo de análise é dependente do Qemu. Isso faz com que o Anubis não funcione com outro tipo de tecnologia de virtualização nem possa ter seu mecanismo de análise instalado em máquinas reais (*bare metal*), o que é uma característica desejável ao sistema proposto neste artigo;
- CWSandbox, por sua vez, não seria funcional, nem poderia ser implementado em sistemas Windows 8 da mesma forma que está atualmente, dado que se baseia em injeção de DLLs. Isto faz com que ele sofra das limitações expostas na Seção 2.3. Além disso, há a necessidade de uma nova injeção de DLL para cada processo criado, tornando o mecanismo de monitoração muito custoso;
- Cuckoo Sandbox, além das restrições contra injeção de DLLs, o fato de o código ser aberto e disponível facilita que os atacantes criem mecanismos de anti-análise e impeçam a monitoração adequada de seus exemplares de *malware*.
- BehEMOT, por ser baseado em um *driver*, é flexível o suficiente para atuar em máquinas virtuais e reais, além de executar em um nível mais privilegiado que a maioria dos exemplares de *malware*. No entanto, dado que o mecanismo de interceptação de chamadas de sistema é implementado por meio da técnica de *SSDT hooking*, seu funcionamento em Windows 8 não é possível (Seção 2.1).
- Capture-BAT, por ser implementado sob a forma de *filter drivers*, permite flexibilidade de atuação, pois exige apenas a instalação do *driver* no sistema *guest* ou

real (ambiente monitorado). Logo, pode operar sobre diversos tipos de sistemas, como *bare metal*, VirtualBox, KVM, VMWARE, entre outros. Além disso, devido ao fato da implementação por *filter drivers* ser um recurso nativo do sistema operacional, esta não sofre das limitações impostas às DLLs, embora exija sua assinatura, como descrito na Seção 2.2.

4. Projeto do Sistema Proposto

Nesta seção, as decisões de projeto são expostas, bem como detalhes da implementação realizada e da arquitetura proposta para o sistema de análise.

4.1. Decisões de Projeto e Implementação

Para o desenvolvimento da ferramenta, deve-se considerar as limitações das ferramentas/sistemas mencionados na Seção 3.1 e relacionadas a *hooks* em *kernel* descritas na Seção 2.1. Rossow et al. [Rossow et al. 2012] estabelece que o método de monitoração deve atuar em um nível mais privilegiado do que o objeto sob análise para minimizar o risco de detecção, sabotagem ou subversão da ferramenta por parte do *malware*, que pode inclusive verificar sua integridade em memória a fim de identificar mecanismos de monitoração (ex.: injeção de DLL feita pela Cuckoo Sandbox). Dessa forma, decidiu-se implementar a ferramenta de análise através de um *driver* de *kernel*, consistindo na mesma abordagem usada na implementação da ferramenta Capture-BAT, porém capturando o tráfego de rede externamente ao ambiente de execução do *malware*. Embora Capture-BAT faça uso apenas da técnica de *filter driver*, deve-se lembrar que sua codificação baseia-se em Windows XP de 32 bits. Portanto, a implementação de uma solução utilizando a mesma técnica em Windows 8 de 64 bits requer a adequação dos tipos de dados (devido ao tamanho da palavra) e dos parâmetros de chamadas de *callback* (Seção 2.4).

Cabe ressaltar que a monitoração feita por Capture-BAT é insuficiente para a avaliação adequada do comportamento exibido por *malware* sob análise, uma vez que não são obtidas certas informações desejáveis, como os valores escritos nas chaves de Registro criadas ou modificadas. Além disso, Capture-BAT é incapaz de monitorar apenas um determinado processo escolhido, interceptando as modificações feitas no sistema operacional como um todo, inclusive as feitas por programas legítimos do próprio sistema que estão em execução em *background*. Na ferramenta de monitoração proposta neste artigo, foi implementado suporte à captura dos valores escritos nos campos alterados e foram desenvolvidos métodos para interceptar apenas o processo submetido para análise e seus processos-filhos ou aqueles cujos processos monitorados interagiram. Logo, concentrou-se esforços na produção de uma solução funcional para Windows 8 com a versatilidade apresentada por Capture-BAT, porém com suas principais limitações resolvidas.

A ferramenta desenvolvida é um *driver* de *kernel* que aplica duas técnicas distintas para interceptação das ações realizadas pelo programa monitorado: ações de **Registro** e de **processos** são obtidas por *callbacks* (Figura 1) e ações do **sistema de arquivos** por um *filesystem filter*. As informações armazenadas em *log* durante a execucão de um dado exemplar de *malware* são: registro temporal (*timestamp*); processo originário da ação (PID e nome); ação realizada (escrita de arquivo, leitura de arquivo, remoção de arquivo, criação de processo, término de processo, definição de chave do Registro ou remoção de chave do Registro); parâmetro/valor da ação; objeto-alvo da ação. Adicionalmente, a **remoção** de arquivos armazena o arquivo a ser removido para análise posterior.

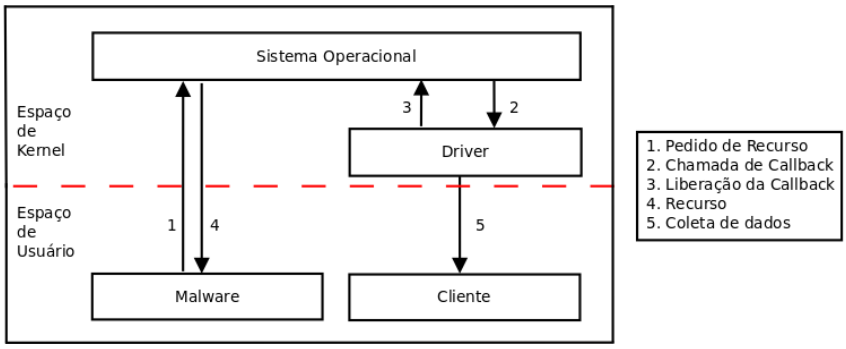

Figura 1. Passos realizados na interceptação por técnica de *callback*.

4.2. Arquitetura do Sistema

A partir do *driver* desenvolvido, projetou-se um sistema automatizado para análise dinâmica com diversos outros componentes auxiliares, como a captura de tráfego de rede e o controle da sua saída. O *driver* provê a comunicação com o nível de usuário de modo que diferentes programas clientes podem obter as informações capturadas através do uso de *I/O Request Packets* (IRP). Um dos componentes implementados é o "CLIENTE", um programa interno ao ambiente da análise que se comunica constantemente com o *driver*, responsável por obter os dados capturados durante a monitoração do *malware*. Este programa também permite alterar configurações básicas do funcionamento do *driver*, como registrar os subsistemas a serem monitorados e ligar/desligar o modo de *debugging*.

Outro componente importante do sistema proposto é o "CONTROLADOR DA ANÁLISE", que consiste de uma aplicação interna ao ambiente de análise, a qual contém suporte a requisições de rede, utilizando-se de um *socket* TCP para obtenção e execução do *malware* via "CONTROLADOR EXTERNO", envio de resultados e recebimento de comandos de controle. Esse componente também é responsável por agrupar os *logs* gerados, o tráfego de rede capturado e os arquivos removidos em um único arquivo a ser transmitido ao ambiente externo.

No âmbito da rede, o tráfego gerado durante a execução do *malware* é capturado externamente ao ambiente da análise e armazenado em formato `pcap` via `tcpdump`. O tráfego de rede capturado durante a execução do exemplar de *malware* é separado do tráfego originado por aplicações do sistema operacional através de filtros elaborados com base na execução de um sistema não contaminado. As boas práticas para a análise de *malware* englobam a execução em ambientes totalmente controlados, de forma a evitar ataques contra terceiros e contaminações. Para tanto, todo o tráfego de saída passa por um *firewall* (`IPTables`) que permite conexões HTTP e HTTPS para que o *malware* possa fazer *download* e verificar a conectividade. outros protocolos. O tráfego relacionado aos demais protocolos de aplicação é redirecionado para um *honeypot* (`honeyd`) para fins de registro dos demais protocolos utilizados. A integração dos componentes em uma arquitetura baseada em máquinas virtuais, para aumentar a escalabilidade, resulta no sistema proposto ilustrado na Figura 2. Cabe ressaltar que o sistema pode atuar em máquinas *bare-metal* sem a necessidade de alteração de código, bastando que se configure o ambiente e seus componentes adequadamente.

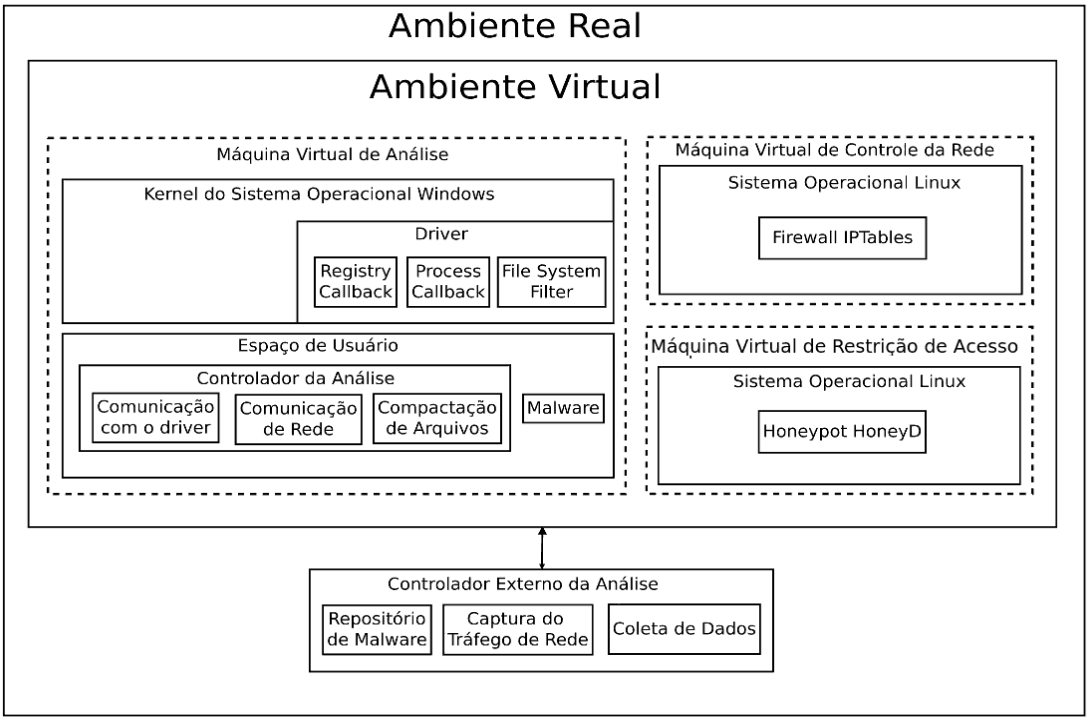

Figura 2. Arquitetura do sistema de análise de *malware* (Windows 8, 64 bits).

5. Testes e Resultados

Nesta seção são apresentados os testes realizados para validar o correto funcionamento do sistema proposto, bem como resultados obtidos da análise. No período entre 01/01/2014 e 21/05/2014 foram coletados 2.937 exemplares de *malware* únicos (com base no *hash* MD5) provenientes de *honeypots*, *phishing* e *downloads* de *links* contaminados. O ambiente de análise definido para os testes consiste de uma máquina virtual com sistema operacional Windows 8 de 64 bits sem qualquer mecanismo de segurança habilitado, de modo a evitar eventuais interferências com o *malware* a ser executado ou com a ferramenta de monitoração desenvolvida. O mecanismo de virtualização utilizado é o Qemu-KVM em operação sobre um *host* Linux Ubuntu 12.04 Server.

5.1. Validação

A fim de verificar se a monitoração das ações efetuadas sobre os subsistemas de arquivos, Registro e processos é feita adequadamente, foram executados alguns exemplares da coleção obtida para este artigo. Após a execução desses exemplares no sistema proposto, trechos foram escolhidos para ilustrar que as operações definidas são monitoradas com sucesso.

A Listagem 1 mostra o exemplar "7G6C5n.exe" definindo o valor "C:\7G6C5n.exe" na chave "...\Windows\CurrentVersion\Run\SoftBrue" através da ação `SetValueKey`. Com isso, o exemplar é executado durante a inicialização do sistema operacional, fazendo com que o *malware* sobreviva ao desligamento da máquina ou a um eventual *reboot*.

Listagem 1. Monitoração de ação de escrita em chave do Registro.

```
1 7/4/2014 -13:3:48.793|SetValueKey|2032|C:\7G6C5n.exe|\REGISTRY\
    USER\S-1-5-21-3760592576-961097288-785014024-1001\Software\
    Microsoft\Windows\CurrentVersion\Run|SoftBrue|"C:\7G6C5n.exe"
```

A Listagem 2 mostra o exemplar "visualizar.exe" escrevendo dados (`WriteOperation`) no arquivo "dll.exe", um programa do sistema. Esse tipo de ação, isto é, a modificação de um arquivo existente, pode causar a inclusão de funcionalidades maliciosas em programas legítimos.

Listagem 2. Captura de ação de escrita no sistema de arquivos.

```
1 7/4/2014 -13:3:48.76|WriteOperation|3028|C:\visualizar.exe|C:\
    Windows\SysWOW64\dll.exe|
```

Já a Listagem 3 mostra o *malware* "deposito.exe" efetuando a remoção do arquivo "rr.txt". Tal arquivo pode ter sido gerado anteriormente pelo exemplar para armazenar alguma informação obtida e a ação `DeleteOperation` indica a remoção de evidências da infecção do sistema-alvo.

Listagem 3. Ação de remoção de arquivo no sistema-alvo.

```
1 7/4/2014 -13:5:1.895|DeleteOperation|2032|C:\deposito.exe|C:\
    ProgramData\rr.txt|
```

A Listagem 4 mostra o programa "visualizar.exe" chamando um programa do sistema modificado anteriormente, como ilustrado na Listagem 2, por meio da criação do processo "dll.exe" (ação `CreateProcess`).

Listagem 4. Processo monitorado devido a interação com malware.

```
1 7/4/2014 -13:3:48.294|CreateProcess|3028|C:\Monitor\Malware\
    visualizar.exe|2440|C:\Windows\SysWOW64\dll.exe
```

A Listagem 5 mostra uma sessão de rede na qual o *malware* acessa a porta 80 (HTTP) de um endereço IP comprometido e efetua a requisição `GET`.

Listagem 5. Exemplo de tráfego de rede capturado durante análise.

```
1 2014-05-14 20:02:40.963113 10.10.100.101 XX.YY.ZZ.121
    HTTP 290 GET /.swim01/control.php?ia&mi=00B5AB4E-47098
    BC3 HTTP/1.1
```

Logo, observa-se que todos os tipos de ações que deveriam ser monitoradas são armazenadas em *logs*, validando o funcionamento do sistema e possibilitando o teste geral com todos os exemplares de *malware* da coleção.

5.2. Resultados com *Malware*

Após verificar que que os resultados produzidos pelo sistema estão corretos, inclusive utilizando programas especialmente feitos para testar suas funcionalidades de monitoração, os 2.937 exemplares coletados foram submetidos para análise dinâmica no sistema proposto. A variedade dos tipos de arquivos nos quais os exemplares encontram-se contidos

é ilustrada na Figura 3. PE (*Portable Executable*) é um formato de arquivo utilizado em sistemas Windows para arquivos executáveis e bibliotecas, entre outros tipos de arquivo, tendo uma versão de 32 bits (PE32) e outra de 64 bits (PE+). DLLs (*Dynamic-Link Libraries*) são bibliotecas compartilhadas no formato PE. Arquivos CPL são um tipo especial de DLL que exportam a função *CPlApplet*. Este tipo de arquivo é usado pelos *applets* do painel de controle do sistema Windows e pode ser executado diretamente por usuários via clique duplo. Mono é uma plataforma de *software* que permite o desenvolvimento de aplicações multi-plataforma. Além disso, ele é uma implementação de código aberto do *framework* .NET da Microsoft.

Todos os 2.937 exemplares de *malware* foram submetidos ao VirusTotal (`http://www.virustotal.com`), um serviço *online* que analisa arquivos com 53 antivírus disponíveis no mercado e fornece as assinaturas de identificação encontradas por cada um deles (rótulos de detecção). Deste total, 2.520 exemplares foram detectados por pelo menos um antivírus na época da submissão, o que alarma pela quantidade de exemplares que podem ter sido usados para infecção de usuários nos primeiros dias de disseminação ($\approx 15\%$). A Figura 4 mostra os 10 rótulos de detecção mais atribuídos ao total de exemplares da coleção.

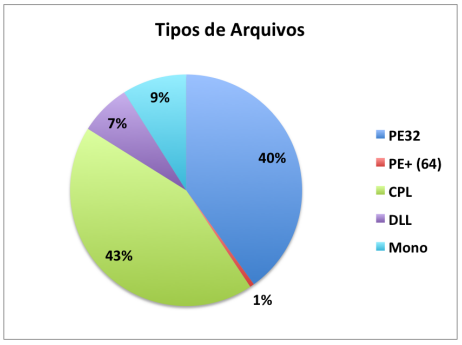

Figura 3. Distribuição de amostras por tipo de arquivo.

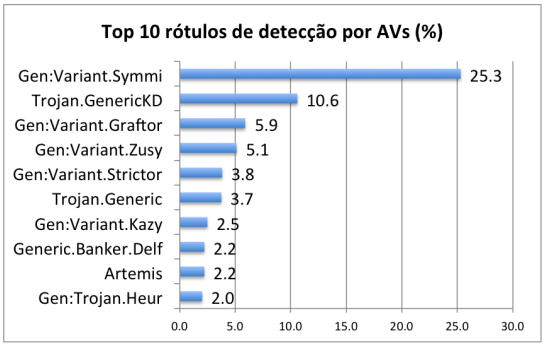

Figura 4. Rótulos de detecção observados (porcentagem).

5.2.1. Comportamentos suspeitos observados no sistema operacional

Na Tabela 1, são mostradas as atividades monitoradas durante a execução dos exemplares de *malware* no sistema proposto neste artigo, bem como quantos dos 2.937 analisados as apresentaram. Deste total, 55 programas não produziram resultados, o que pode ocorrer por diversas razões, tais como não ter acesso a algum componente necessário para a continuidade da execução, o arquivo estar corrompido ou realizar alguma ação não permitida no Windows 8, a identificação da execução em ambiente emulado, entre outras.

A análise mais aprofundada dessas atividades revelou os seguintes comportamentos suspeitos típicos de uma infecção:

- Finalização de mecanismos antívirus instalados no sistema operacional;
- Desligamento do *firewall* nativo do Windows;
- Criação de novos binários no sistema, seja por *download* ou por *dropping*;
- Desligamento do mecanismo de atualização automática do Windows;

Tabela 1. Atividades monitoradas e quantidade de exemplares que as exibiram.

Atividade	Qtde.
Escrita no Registro	1073
Remoção de chave(s) do Registro	772
Criação de processo(s)	602
Término de processos	1337
Escrita em arquivo(s)	1028
Leitura de arquivo(s)	1694
Remoção de arquivo(s)	551

- Tentativa de persistência (sobrevivência a desligamentos e reinicializações);
- Injeção de *Browser Helper Objects* no Internet Explorer;
- Modificação no arquivo `hosts.txt` do sistema operacional;
- Sobrescrita de um arquivo (programa ou biblioteca) já presente no sistema;
- Remoção de seu próprio programa ou de outros artefatos.

5.2.2. Comportamentos suspeitos observados no tráfego de rede

A análise do tráfego de rede capturado durante a execução dos exemplares traz uma perspectiva adicional para o entendimento da atuação do *malware* no sistema infectado. A Tabela 2 mostra os protocolos e portas que mais foram utilizadas pelos exemplares analisados. Nota-se que quase metade deles fazem uso do protocolo HTTP para buscar novos componentes ou enviar dados para o atacante, uma vez que a porta 80 geralmente não é bloqueada na saída. Também é interessante ressaltar as atividades das outras portas, obtidas após análise manual do tráfego capturado: a porta 9000 foi utilizada com destino de comunicação similar à de *bots*, recebendo dados com um formato parecido com JSON; a porta 2869 foi utilizada para troca de tráfego HTTP; nenhuma tentativa de comunicação com a porta 720 teve sucesso no fechamento do *3-way handshake*; a porta 82 foi utilizada tanto para recebimento de tráfego HTTP como para tráfego aparentemente codificado; a porta 8181 foi utilizada no recebimento de informações evadidas do sistema-alvo, sem prover resposta do lado do servidor.

Tabela 2. Top 10 Protocolos/portas mais utilizados por *malware* (% do total de exemplares) observados no tráfego de rede capturado durante a análise.

Proto	HTTP	HTTPS	MS-SQL	-	SMTP	-	MySQL	-	-	-
Porta	80	443	1433	8181	587	82	3306	720	2869	9000
Qtde.	44,4%	6,5%	2,6%	1,0%	0,8%	0,7%	0,5%	0,3%	0,3%	0,2%

A análise mais aprofundada do tráfego de todos os exemplares em busca de comportamentos que indicam a presença de códigos maliciosos produziu os resultados apresentados na Tabela 3. Tais comportamentos incluem: *Download* desconhecido, isto é, binários executáveis não identificados por mecanismos antivírus; *E-mail/Spam*, que consiste do envio de informações por *e-mail* ou tentativa de envio de *spam*; *Banker*, que indica *malware* que tenta evadir credenciais do usuário (agência, conta, tabela de senhas, senha

do *Internet Banking*); comunicação IRC, na qual são identificados comandos típicos de protocolos de *Instant Relay Chat*; dados do sistema (nome, usuário, versão) evadidos via rede; obtenção de PAC (*Proxy Auto Configuration files*), arquivos carregados no *browser* que modificam a navegação; portas de IRC, que indica que uma porta comumente associada a este tipo de protocolo foi acessada.

Tabela 3. Comportamentos suspeitos observados no tráfego de rede.

Comportamento	Qtde. de *malware*
Download desconhecido	154
E-mail/Spam	25
Banker	22
Comunicação IRC	4
Dados do sistema	3
Obtenção de PAC	1
Portas de IRC	1

Discussão. O sistema proposto é o único de que se tem notícia que é tanto capaz de executar arquivos no formato PE+ (64 bits) quanto de prover um ambiente de 64 bits (Windows 8) para análise de *malware*. Exemplares de 64 bits foram submetidos para os sistemas de análise dinâmica *Anubis* (`http://anubis.iseclab.org`), *Cuckoo*, *ThreatExpert* (`http://www.threatexpert.com`), Camas Comodo (`http://camas.comodo.com`) e CWSandbox (`http://www.threattracksecurity.com/resources/sandbox-malware-analysis.aspx`) em suas versões disponíveis publicamente. Destes, nenhum foi capaz de realizar a análise, seja por não suportar explicitamente o tipo de arquivo ou por não retornar resposta, indicando um *crash* no sistema. Um ponto interessante sobre os exemplares analisados diz respeito aos rótulos de detecção providos pelos antivírus: a maioria deles baseia-se em heurísticas genéricas que, embora permitam que o usuário seja alertado sobre um programa malicioso, não provêem informações sobre o tipo de dano causado. Um sistema de análise dinâmica como o proposto neste artigo complementa uma ferramenta antivírus, provendo informações do comportamento, além de permitir a identificação de programas suspeitos quando ainda não há assinaturas ou heurísiticas codificadas. A constatação mais grave acerca dos resultados obtidos é que, mesmo com os mecanismos de segurança propostos a partir do NT 6, a retrocompatibilidade faz com que exemplares de 32 bits codificados e compilados para Windows XP infectem também os Windows 8 caso o atacante subverta o sistema operacional e desabilite tais mecanismos.

6. Conclusão

Neste artigo, introduziu-se o projeto de arquitetura e a implementação de um sistema de análise dinâmica de *malware* de 64 bits baseado em Windows 8, com suas características, desafios e decisões tomadas. O funcionamento do sistema, único do tipo do qual se tem notícia, foi avaliado por meio da execução de 2.937 exemplares de *malware*, cujos resultados mostraram a utilidade da monitoração das ações no nível da rede e do *kernel* do sistema operacional para a identificação de comportamentos suspeitos. Os resultados obtidos permitem uma maior compreensão da atuação de *malware*, possibilitando a criação de heurísticas de detecção, procedimentos de remediação e tomada de contra-medidas

para resposta a incidentes. Os trabalhos futuros incluem a integração do ambiente *bare-metal* ao ambiente emulado a fim de se monitorar *malware* que possua mecanismo anti-análise, a implementação de técnicas para monitoração de outros subsistemas (como o de gerenciamento de memória) e o estudo e desenvolvimento de mecanismos de proteção para a ferramenta de monitoração, visando evitar a detecção por parte de *malware* mais complexo e consequente evasão da análise.

Referências

Bayer, U., Kruegel, C., and Kirda, E. (2006). Ttanalyze: A tool for analyzing malware. In *15th European Institute for Computer Antivirus Research (EICAR 2006) Annual Conference*.

Bellard, F. (2005). Qemu, a fast and portable dynamic translator. In *Proceedings of the Annual Conference on USENIX Annual Technical Conference*, ATEC '05, pages 41–41, Berkeley, CA, USA. USENIX Association.

Filho, D. S. F., Grégio, A. R. A., Afonso, V. M., Santos, R. D. C., Jino, M., and de Geus, P. L. (2010). Análise comportamental de código malicioso através da monitoração de chamadas de sistema e tráfego de rede. *Anais do X Simpósio Brasileiro em Segurança da Informação e de Sistemas Computacionais*.

Guarnieri, C. (2013). Cuckoo sandbox. `http://www.cuckoosandbox.org/`. Acesso em junho/2014.

Microsoft (2013a). CreateRemoteThread. `http://msdn.microsoft.com/en-us/library/windows/desktop/ms682437(v=vs.85).aspx`. Acesso em junho/2014.

Microsoft (2013b). Detours. `http://research.microsoft.com/en-us/projects/detours/`. Acesso em junho/2014.

Microsoft (2013c). Kernel patch protection for x64-based operating systems. `http://technet.microsoft.com/pt-br/library/cc759759(v=ws.10).aspx`. Acesso em junho/2014.

Microsoft (2014a). CmRegisterCallback. `http://msdn.microsoft.com/en-us/library/windows/hardware/ff541918(v=vs.85).aspx`. Acesso em junho/2014.

Microsoft (2014b). CmRegisterCallbackEx. `http://msdn.microsoft.com/en-us/library/windows/hardware/ff541921(v=vs.85).aspx`. Acesso em junho/2014.

Rossow, C., Dietrich, C. J., Kreibich, C., Grier, C., Paxson, V., Pohlmann, N., Bos, H., and van Steen, M. (2012). Prudent Practices for Designing Malware Experiments: Status Quo and Outlook . In *Proceedings of the 33rd IEEE Symposium on Security and Privacy (S&P)* , San Francisco, CA.

Seifert, C., Steenson, R., Welch, I., Komisarczuk, P., and Endicott-Popovsky, B. (2007). Capture - a behavioral analysis tool for applications and documents. *Digital Investigation*, 4S:S23–S30.

Willems, C., Holz, T., and Freiling, F. (2007). Toward automated dynamic malware analysis using cwsandbox. *IEEE Security & Privacy*, 5:32–39.