DOI: 10.5769/C2015008 or http://dx.doi.org/10.5769/C2015008

Maldetect: Uma metodologia automatizável de detecção de *malwares* desconhecidos

Leandro Silva dos Santos, Dino Macedo Amaral

***Resumo*— O cenário de ataques cibernéticos está atingindo níveis cada vez mais altos de complexidade. Com isso as ferramentas tradicionais de detecção e remoção de ameças estão cada vez menos eficiente, pois utilizam um abordagem de detecção baseada em assinatura. Este trabalho propõe uma metodologia automatizável de detecção de *malwares* desconhecidos, ou seja, aqueles que não foram detectados pelas ferramentas tradicionais. A metodologia *Maldetect* apresentada neste artigo coleta e correlaciona características comportamentais típicas de códigos maliciosos, sendo independente de sistema operacional. Foi construída uma ferramenta usando as linguagens de programação PHP e Python, denominada *Maldetect Tool* que automatiza a metodologia proposta. A partir do *dump* da memória volátil, a *Maldetect Tool* gera um relatório contendo os processos que mais realizam atividades típicas de *malwares*. A *Maldetect Tool* analisou de maneira automatizada *dumps* de memória de estações infectadas e foi capaz de detectar os artefatos maliciosos a partir da análise da memória volátil.**

***Palavras-Chave*— Forense, análise de memória, detecção de *malwares* desconhecidos, volatility, maldetect**

***Abstract*— The scenario of cyber attacks is reaching ever higher levels of complexity. Thus the traditional tools of threats detection are becoming less efficient because they use mainly signature-based detection. This work proposes a automatable methodology of unknown malware detection, ie those that were not detected by traditional tools. The Maldetect methodology presented in this paper collects and correlates typical behavioral characteristics of malicious code and is independent of operating system. A so-called Maldetect Tool that automates the proposed methodology was built using Python and PHP programming languages. From the dump of volatile memory, Maldetect Tool generates a report containing the processes that perform more typical activities of malware. The Maldetect Tool analyzed in an automated approach memory dumps from infected machines and was able to detect malicious artifact from the analysis of volatile memory.**

***Keywords*— Forensics, memory analysis, unknown malwares detect, volatility, maldetect**

I. Introdução

O cenário de ataques cibernéticos acompanha a modernização das ferramentas de detecção e remoção, o que os tornam cada vez mais complexos. A indústria de antivírus (AV) tem se demonstrado ineficiente contra ameaças avançadas, principalmente por utilizar detecção baseada em assinaturas. Este tipo de detecção é facilmente burlado com técnicas de polimorfismos e metamorfismo[2]. Apesar disso, o AV ainda possui seu espaço no arsenal de segurança da informação.

Leandro Silva dos Santos, Dino Macedo Amaral. Departamento de Engenharia Elétrica, Universidade de Brasília, Brasília-DF, Brasil, E-mails: holyminds@gmail.com, dinoamaral@gmail.com

Inicialmente, os *malwares* tinham objetivos simples, como apagar arquivos ou provocar erros, ou ainda executar atividades indesejadas em um computador, as quais eram percebidas pelos usuários. Porém, com o avanço dos *malwares*, os mesmos são capazes de capturar e até sequestrar dados relevantes das vítimas. Este último tipo de *malware* (chamados de *rasomware*[1]) criptografa os arquivos da vítima e pedem pagamento pela decifração destes arquivos [3].

Existe ainda o conceito de *Advanced Persistent Threats* - APT (Ameaça Persistente Avançada), a qual geralmente possui alvos específicos e utiliza técnicas avançadas, como a exploração de uma ou mais vulnerabilidades *0-day* e o uso de certificados falsificados, para comprometer as estações de seus alvos[21]. Assim, na maior parte dos casos, não são produzidos por indivíduos isolados, mas sim por instituições, crime organizado ou governos que mediante objetivos específicos ajudam a financiar tais atividades[3]. Este tipo de ameaça geralmente é usada em atividades de espionagem ou sabotagem [10].

Diante deste cenário, a análise de memória volátil consiste em umas das principais técnicas para analisar ameaças avançadas, por ser eficiente na identificação de características comportamentais típicas de *rootkits* e outros tipos de *malwares*[18]. Além disso, a análise de memória permite reconstruir o estado original do sistema, quais arquivos estão sendo acessados, as conexões de redes que foram abertas, dentre outros dados relevantes para a identificação de código malicioso[11].

Dessa forma, este trabalho propõe uma metodologia automatizável de análise de memória volátil, denominada *Maldetect*. Esta é capaz de coletar caraterísticas comportamentais e correlacionar as informações de forma a identificar quais são os processos candidatos a *malware*. Além da metodologia, é apresentada uma implementação da Maldetect que automatiza a análise do *dump* de memória volátil do sistema operacional Window 7, bem como os resultados obtidos dessa análise.

Este trabalho está organizado da seguinte forma: a seção II descreve os trabalho relacionados e mostra algumas soluções de automatização de análise de memória volátil que já foram propostas. A seção seguinte apresenta três metodologias de análise de memória para detecção de artefatos maliciosos. A seção IV descreve a ferramenta Volatility usada para implementação da automatização da Maldetect. A seção V detalha as fases da metodologia Maldetect para detecção de *malwares* desconhecidos a partir da análise do *dump* de memória volátil. A seção VI descreve as técnicas utilizadas na construção de uma ferramenta que implementa a metodologia Maldetect para o sistema operacional Windows 7. A seção VII apresenta os resultados obtidos a partir da execução da Maldetect para detecção de artefatos maliciosos em *dumps* de memória volátil de estações infectadas e por fim, na última

seção, são feitas as considerações finais.

II. Trabalhos Relacionados

Análise de memória foi um dos principais temas do desafio de 2005 do Digital Forensic Research Workgroup (DFRWS), o que motivou um esforço de pesquisa e desenvolvimento de ferramentas nesta área[4]. Este desafio deu início aos estudos de análise de memória usando técnicas forenses. Em [20], no ano de 2011, Vömel publicou um *survey* que apresenta várias técnicas de aquisição de memória baseada em software e hardware. Mostrando também várias técnicas de análise de processos, de recuperação de chave criptográfica, de análise de registro de sistemas, de redes, de arquivos, do estado do sistema e de uma aplicação específica baseada nas estruturas de memória do sistema operacional. Essas técnicas são amplamente utilizadas em análise manual de memória.

Em [7], no ano de 2013, Liang Hu et al. mostrou a importância de automatizar o processo de análise de memória volátil. O artigo propõe automatizar a análise de memória baseada em dois fluxos de análise (*DLL flow* e *Process flow*). Em cada fluxo são coletados diversos dados que serão processados e correlacionados gerando um relatório. Porém apenas os dados do fluxo de DLL são processados automaticamente.

Em [5], no ano de 2014, outra solução de automatização da análise de memória foi apresentada por Fahad - *Associate Director – Security Research and Analytics UBS AG*. Ela é composta de três fases: primeiro aquisição da memória para um *drive* seguro que fique oculto para o usuário. Segundo, é a execução do Volatility para extração das informações relevantes contidas no *dump* da memória. As duas primeiras fases serão executas a cada 30 minutos. Por fim, essas informações são enviadas para um servidor central que fará a análise. Esta fase executará um algoritmo de comparação do *dump* atual com as informações contidas na base de dados. Assim é possível identificar a criação de novas conexões de redes, novos serviços, alteração e criação de chaves de registros, entre outros dados. O problema é o aumento do processamento do *host* e o volume de dados sendo transferidos pela rede.

Como a análise de memória tem sido amplamente aplicada na identificação de código malicioso, existem alguns cuidados que devem ser levados em consideração na fase de coleta da memória volátil, pois o processo de aquisição, geralmente, requer a execução de código na máquina infectada. Este processo pode ser interferido pelo *malware* em execução. Em [18], no ano de 2013, Johannes analisou várias técnicas antiforense que interferiam na aquisição da memória e testou as principais ferramentas com o objetivo de identificar quais delas seriam resistentes a estas técnicas. O resultado encontrado pode ser observado na figura 1 e nenhuma ferramenta foi resistente a todas as principais técnicas antiforense.

Dessa forma, este trabalho propõe uma metodologia automatizável de detecção de *malwares* desconhecidos[1] baseada em características comportamentais dos processos, DLLs, e drivers em execução no sistema operacional. O dados serão extraídos da imagem da memória volátil e correlacionados com outras fontes, como o VirusTotal[2] e *blacklist* de IP's. Além disso, a metodologia é independente de sistema operacional.

[1] *Malwares* que não foram detectados pelas ferramentas tradicionais de detecção de códigos maliciosos

III. Metologias de Análise de Memória

No campo da computação forense, a análise de memória pode trazer resultados mais proveitosos que a análise de artefatos de disco, já que a análise de memória identifica as ações que estão sendo executadas pelo sistema operacional e pelos aplicativos em execução no momento da coleta do *dump* da memória volátil. Além disso, a análise de memória pode prover várias informações sobre o estado do sistema em tempo de execução, por exemplo: quais processos estão em execução, conexões de rede abertas e comandos executados recentemente. Os dados que ficam criptografados no disco, geralmente não estão criptografados quando executados na memória. Também é possível encontrar na memória chaves criptográficas, arquivos confidenciais e histórico dos *browsers* no modo de navegação anônima [12].

Dessa foram, a seguir são descritas três metodologias de análise de memória volátil para detecção de artefatos maliciosos, as quais constituirão a base da metodologia proposta neste trabalho.

A. Metodologia do SANS - System Administration, Networking and Security

A metodologia do SANS de análise de memória está focada em busca de artefatos maliciosos residentes em memória, ou seja, em execução no sistema operacional. Sua descrição não está concentrada em um único documento, mas é descrita em alguns casos de uso e *posters* públicos. Em [9] a metodologia é apresentada como sendo o nono passo da busca por um *malwares* desconhecidos, como mostra a Figura 2. A metodologia proposta pelo SANS é composta de seis passos, alguns desses passos já são executados em uma análise padrão de memória, e outros são específicos para encontrar artefatos maliciosos. A análise de memória volátil nos fornece melhores resultados para identificar técnicas usadas por *rootkits*, os quais procuram dificultar sua detecção.

- **Identificar processos estranhos**

  Na fase de análise de processos devemos coletar algumas informações, tais como: nome do processo, caminho em disco, processo "pai", linha de comando, hora de inicialização e SIDs. Esses dados serão usados para: identificar processos legítimos; verificar a escrita correta do nome; identificar caminhos suspeitos dos processos; verificar o "pai" do processo; e identificar parâmetros de linha de comando que iniciou o processo.

- **Analisar DLLs e handles[3] de processos**

  Existe uma diferença fundamental entre programa e processo. Um programa é uma sequencia estática de

[2] VirusTotal é um serviço gratuito que analisa arquivos e URL's suspeitas e facilita a rápida detecção de vírus, worms, cavalos de tróia e todos os tipos de *malwares*. Acesso em: https://www.virustotal.com/

[3] Referência abstrata para um recurso[16].

Acquisition tool	Version	Format	KDBG	MmGetPhysical memory-ranges()	MmMap-MemoryDump-Mdl()
Memoryze	2.0	raw	PASS	FAIL	PASS
FTK Imager	3.1.2	raw	PASS	FAIL	PASS
Win64dd	1.4.0	raw	PASS/FAIL	FAIL	FAIL
Win64dd	1.4.0	dmp	FAIL	FAIL	FAIL
DumpIt	1.4.0	raw	PASS	FAIL	FAIL
WinPmem	1.3.1	raw	FAIL	FAIL	PASS
WinPmem	1.3.1	dmp	FAIL	FAIL	PASS
WindowsMemoryReader	1.0	raw	PASS	FAIL	PASS
WindowsMemoryReader	1.0	dmp	PASS	FAIL	PASS

Fig. 1. Resultado da aquisição de memória com técnicas de antiforense ativada[18].

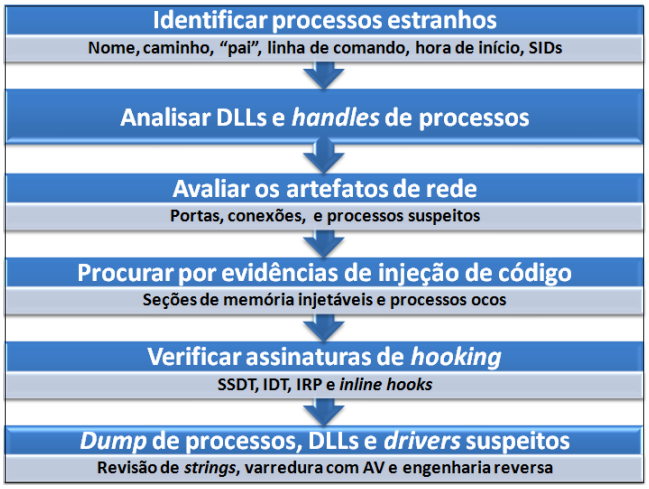

Fig. 2. Metodologia de análise de memória do SANS, adaptado de [9].

instruções; já um processo contém um conjunto de recursos usados por uma instância de um programa. Em [16] são apresentados componentes de um processo do sistema operacional Windows:

* Um espaço de endereçamento virtual;
* Um programa executável;
* Um lista de *handles* para vários recursos do sistema (por exemplo: portas de comunicação, semáforos e arquivos abertos);
* Uma lista de *Threads*;
* Um contexto de segurança; e
* Lista de DLLs (*dynamic-link libraries*) associadas.

Nesta fase são analisados os *handles* associados aos processos de maneira a identificar alguma atividade maliciosa, como por exemplo DLLs que apesar de terem nomes legítimos estão em caminhos diferentes no sistema de arquivos do sistema operacional. Além disso, muitos *malwares* usam um *handle* do tipo *mutex*[4] para identificar se o *malware* já foi instalado na máquina da vítima e assim não executar nada.

- **Avaliar os artefatos de rede**

Durante a análise dos artefatos de rede deve-se identificar as portas TCP suspeitas e os processos associados a elas, bem como os indicativos da presença de *backdoors* e a reputação dos IP's que a máquina está conectada. Dessa forma, podemos identificar atividades tipicas de códigos malicioso com relação as conexões de rede[19].

- **Procura por evidências de injeção de código**

Em [19], são apresentadas duas técnicas de injeção de código:

* *Injeção de DLLs*: muito utilizada pelos *malwares*, eles utilizam algumas chamadas de sistema, tais como: *VirtualAllocEx(), CreateRemoteThread e SetWindowsHookEx()* para carregar uma DLL em um processo já em execução; e
* *Processos ocos*: o *malware* cria uma nova instância de um processo legítimo e substitui a área de código do processo legítimo pelo seu código malicioso e obtêm as DLLs, os *handles* e outros recursos do processo original.

A detecção de injeção de código pode ser feita varrendo a memória procurando setores marcados com permissão de escrita e execução e que não tenham um mapeamento para um arquivo. Também pode ser feita uma comparação entre o código de memória e código do arquivo em disco para verificar o nível de similaridade[19].

- **Verificação de assinaturas de *hooking***

Basicamente existem quatro técnicas utilizadas pelos *rootkits* que devem ser verificadas nesta fase da metodologia. Deve-se verificar a *System Service Descriptor Table* (SSDT) - tabela que contém um *array* de ponteiros para as funções de tratamento de cada *system call*. As entradas da SSDT podem ser alteradas pelos *rootkits* e assim alterar a saída ou a entrada das chamadas de sistema, de forma a esconder processos, arquivos e chaves de registros [16].
Além disso, *drivers* maliciosos podem utilizar a *Interrupt Descriptor Table* - estrutura de dados que armazena os endereços das funções de tratamento de interrupção e exceções de processos - para realizar um *hook*[5] em todas

[4]Mecanismo de sincronização usado para serializar o acesso à um recurso [16].

[5]Técnica utilizada por *rootkits* para modificar o comportamento normal de uma ação do sistema operacional ou processo.

rotinas de tratamento ou em apenas um ponto [11].
Outra técnica utilizada é o *hook driver* (*Hooking the I/O Request Packet* - IRP), na qual os *rootkits* alteram a IRP - estrutura de dados que contém códigos para identificar as operações desejadas e *buffers* de dados que serão lidos ou escritos pelo *driver*. Geralmente, o módulo tcpip.sys é atacado com esta técnica[11].
Por último, pode ser usada a *inline hook*, também conhecida como *Dynamic code patching*, que sobrescreve os primeiros *bytes* de um função com a instrução de *jump* (instrução JMP do *assembly* [8]) para redirecionar a execução para a função do código malicioso, e ao final de sua execução retornar para a função original [15].

- **_Dump_ de processos, DLLs e _drivers_ suspeitos**

  Nesta fase, espera-se obter uma lista dos possíveis artefatos malicioso que precisam de uma análise mais profunda. Para esses possíveis *malwares* deve-se realizar um *dump* do processo correspondente da memória e realizada uma revisão de strings, varredura com antivírus, engenharia reversa e outras técnicas que possibilitem a detecção de atividades maliciosas.

B. Caçando Malwares nos processos em memória

Em [12], Michael Hale et al. descreve sete objetivos da análise de memória para se encontrar um *malware*, que são:

- **Recuperar linhas de comandos e caminho dos processos**

  O *Process Enviroment Blobk* (PEB), que é membro da estrutura de memória _EPROCESS, contém o caminho completo do processo, a linha de comando que iniciou o processo, ponteiros para a *heap* do processo, entre outras informações. Essas informações ajudam a localizar o arquivo executável no disco e descobrir informações sobre como o processo foi instanciado na memória da estação infectada.

- **Analisar _heaps_**

  Os dados que as aplicações manipulam (dados recebidos via rede, ou textos digitados em um processador de texto) possuem uma boa chance de estarem armazenados na *heap* do processo, assim não perde-se muito tempo pesquisando em regiões de memória que contém DLLs, arquivos mapeados e a pilha.

- **Inspecionar variáveis de ambiente**

  Existem famílias de *malwares* que marcam sua presença com a criação de variáveis de ambiente. Outros *malwares* manipulam os valores das variáveis de ambientes para gerar comportamentos maliciosos em outros processos. Algumas variáveis que são tipicamente manipuladas por códigos maliciosos, são:
  * PATH: armazena o caminho dos executáveis;
  * PATHEXT: extensões atribuídas aos programas executáveis;
  * Caminho dos diretórios temporários;
  * Caminho dos diretórios de documentos, histórico de internet e dados de aplicações dos usuários; e
  * ComSpec: localização do cmd.exe;

- **Detectar _backdoors_ com _handles_ padrões**

  Identifique se a entrada e saída de um processo estão sendo direcionados a um *socket* de rede remoto para um atacante. Uma técnica muito comum, usada pelos *backdoors* é criação de um *socket* de rede associado a um processo cmd.exe de tal forma que toda saída do processo seja transmitido pela rede e toda entrada do *socket* seja transformada em entrada para o processo.

- **Enumerar DLLs**

  Os *Dynamic Link Libraries* possuem códigos e recursos que podem ser compartilhados entre processos, por isso é muito comum entre os *malwares* a técnica de injetar DLLs em processos legítimos. Durante a análise de DLLs deve-se verificar se existe alguma não vinculada, se o caminho das DLLs no sistema de arquivos são adequados e o contexto em que as mesmas estão carregadas.

- **Extrair arquivos PE da memória**

  Pode ser realizado o *dump* do conteúdo em memória dos programas executáveis para uma análise mais profunda deste artefato. Porém um processo ao ser carregado na memória sofre algumas alterações que devem ser levadas em consideração durante a análise do artefato extraído. Por exemplo, o *hash* md5 do *dump* do processo extraído da memória pode não ser o mesmo do *hash* do arquivo no disco, mas é possível usar um *fuzzy hash*[22] para determinar o grau de similaridade.

- **Detectar injeção de código**

  São apresentados quatro técnicas de injeção de código:
  * Injeção remota de DLLs: o processo malicioso força o processo alvo a carregar uma DLL específica;
  * *Injeção remota de código*: o processo malicioso escreve código na área de memória do processo alvo e força sua execução;
  * *Injeção reflexiva de DLL*: o processo malicioso escreve o código da DLL no espaço de memória do processo alvo; e
  * *Injeção em processo oco*: o processo malicioso inicia uma nova instância de um processo legítimo em modo suspenso e então é feita uma sobrescrita da área de código do processo legítimo pelo código malicioso e sua execução é iniciada.

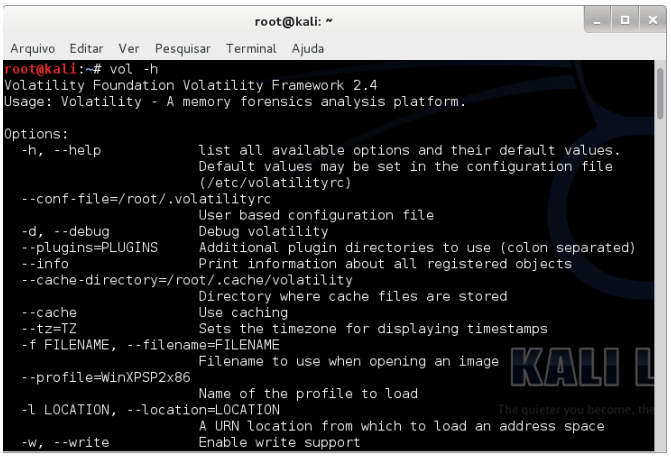

Fig. 3. Volatility em linha de comando.

C. *Metodologia de análise de memória*

Em [14], são descritos os objetivos da análise de memória, especificamente no contexto de análise de código malicioso:

- Coletar os metadados disponíveis, tais como: detalhes de processos, conexões de rede, e outras informações associadas ao potencial *malware*;
- Para cada processo de interesse, se possível, recuperar o arquivo executável da memória para análise; e
- Para cada processo de interesse extrair mais dados da memória, por exemplo, usuários, senhas e chaves criptográficas.

IV. *Volatility Framework*

O *Volatility Framework* é uma coleção de ferramentas, implementada em Python, capaz de extrair artefatos digitais de um *dump* da memória volátil (RAM). O Volatility é licenciado pela *GNU General Public License 2*, possui código aberto e é gratuito. Sua arquitetura permite a inclusão de novas funcionalidades através da criação de novos *plugins* [12].

O Volatility é capaz de analisar o *dump* de memória das versões 32-bits e 64-bits dos sistemas operacionais Windows, Linux, Mac e 32-bits do Android. O Volatility suporta a inclusão de novos sistemas operacionais devido a sua arquitetura modular. Porém o volatility não é uma ferramenta de aquisição de memória e também não possui interface gráfica, seu uso é através de linha de comando [12], como mostra a Figura 3.

Em [6], são apresentados os *plugins* do volatility, os quais são agrupados na seguintes categorias:

* *Image Identification*: identificação do sistema operacional e suas estruturas de dados;
* *Processes and DLLs*: lista os processos e DLLs carregadas na memória;
* *Process Memory*: recupera informações específicas de um ou mais processos;
* *Kernel Memory and Objects*: lista e verifica componentes do *kernel*;
* *Networking*: recupera atividades de rede do *dump* da memória;
* *Registry*: recupera dados armazenados nos registros do sistema operacional;
* *Crash Dump, Hibernation e Conversion*: executa o *parser* e analisa informações de arquivos de hibernação e *crash dumps*, bem como realiza a conversão entre esse tipos de arquivos; e
* *Miscellaneous*: agrupa os *plugins* de tipos diversos.

V. Metodologia Maldetect

A metodologia proposta é uma adaptação da metodologia do SANS descrita na seção III. Esta metodologia é independente de sistema operacional, ou seja, os conceitos podem ser aplicados a qualquer S.O. A Maldetect coleta e correlaciona informações comportamentais dos artefatos residentes no *dump* de memória volátil e identifica quais dessas características são típicas de códigos maliciosos. A figura 4 apresenta um resumo de cada fase e a seguir estas serão descritas em detalhes.

A. ***Pré-análise***

A pré-análise é uma fase de preparação e otimização da metodologia. Possui como objetivo reaproveitar o conhecimento aprendido das execuções anteriores da metodologia. Nesta fase, o *dump* de memória a ser analisado é comparado com a base de conhecimento de indicativos de comprometimentos[6] (IOC) de análises já realizadas, caso não exista uma base de conhecimento prévia, esta etapa poderá ser suprimida. Além disso, nesta etapa pode ser criada uma linha base com dados de atividades consideradas normais para máquina que será analisada.

B. ***Análise de processos***

Deve-se possuir uma lista dos processos do núcleo do sistema operacional a ser analisado, assim como as atividades normais que estes processos possuem. As características desses processos devem ser confrontadas com os processos correspondentes do *dump* da memória em análise de forma a identificar as possíveis anomalias e armazená-las para serem correlacionadas com os resultados das outras fases. Para realizar esta atividade devem ser coletados os seguintes dados de todos os processos em execução: nome, caminho completo, PID, PPID, linha de comando de inicialização, hora de inicialização, hora de término, processos filhos, prioridade de execução, dono do processo, sessão em que está rodando, número de *threads* em execução e número de *handles*. Ainda nesta fase, devem ser usadas várias técnicas de listagem de processos em execução, com o objetivos de encontrar aqueles que usam técnicas de ocultação, ou seja, processos que apesar de estarem em execução não seriam listados no gerenciador de tarefas do sistema operacional. Por fim, deve-se verificar se os binários estão sendo executados a partir de pastas temporárias e se existem processos com nomes similares ao processos do núcleo do sistema operacional.

[6]Descrição das atividades maliciosas que caracterizam determinado *malware*

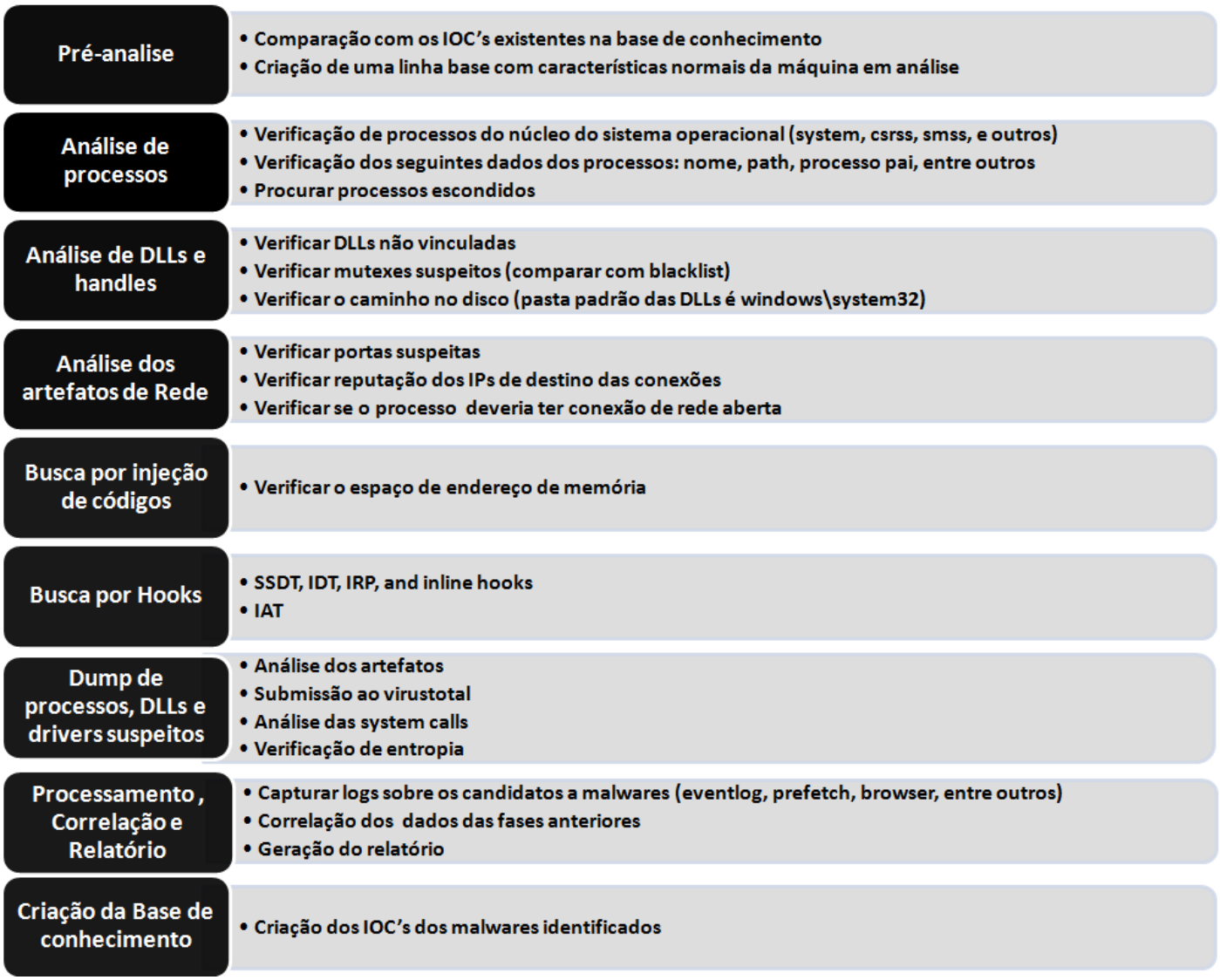

Fig. 4. Metodologia Maldetect

C. *Análise de DLLs e handles*

Durante a análise de DLLs deve-se verificar a existência de DLLs não vinculadas, o contexto onde as mesmas foram carregadas, o caminho do disco onde as DLLs estão armazenadas e a escrita correta de seus nomes. Além disso, os *handles* do tipo *mutexes* devem ser comparados com uma *blacklist* de nomes usados por *malwares* conhecidos. Geralmente este é um recurso muito utilizado pelos códigos maliciosos para identificar se a máquina já foi infectada. Também, verifique se existe um *pipe* (mecanismo que redireciona a saída de uma programa como entrada de outro) redirecionando entradas e saídas do processo "cmd.exe" para um programa remoto, pois esta técnica é muito usada por *backdoors*.

D. *Análise de Artefatos de Rede*

O acesso a rede é muito utilizado pelos *malwares* para diferentes fins, tais como: extravio de informações, *download* de novos componentes, comunicação com a central de comandos, infecção de novas vítimas, disparo de envio de e-mails em massa e ataques de negação de serviços. Dessa forma, deve-se listar as portas abertas em modo *listening* do protocolo TCP e verificar quais portas são consideradas como suspeitas. As informações coletadas na pré-análise pode ajudar a identificação destas portas suspeitas, pois caso a máquina em análise seja um servidor FTP, a porta 21 estará marcada como normal, porém caso estas informações não sejam conhecidas previamente deve-se considerar as portas que não pertencem ao funcionamento normal do sistema operacional como anomalias a serem correlacionadas na fase apropriada. Além disso, deve-se realizar uma verificação do IP's de destino de conexão com uma *blacklist* de IP's maliciosos. E por fim, verifique se existem interfaces de rede em modo promíscuo, quais processos estão fazendo uso das conexões de rede e se estes deveriam fazer uso deste recurso.

E. *Busca por injeção de código*

As técnicas usadas nesta etapa deve ser capaz de identificar as assinaturas de injeção de código, tais como a injeção

de DLLs, verificando áreas de memória marcadas como READ/WRITE/EXECUTE. Além disso, deve ser capaz de verificar se existem *Hollow Process*, esta técnica está descrita na seção III.

F. *Busca por hooks*

O objetivo dessa etapa é identificar os artefatos que usam técnicas avançadas para dificultar sua detecção. As técnicas usadas nesta fase devem ser capazes de identificar os principais tipos de *hooks* usados pelo *malwares*. Deve-se identificar a existência de módulos não vinculados, *hooks* de *system calls* (SSDT), *inline hooks*, alterações na *Interrupt Descriptor Table* (IDT), nos *handles* da *I/O Request Packets* (IRP). Além disso, devem ser identificados os *hooks* da *Import Address Table* (IAT). Também, identifique quais processos estão executando em modo *debug* e a existência de *threads* órfãs. Essa técnicas foram descritas na subseção III-A.

G. *Dump de processos, DLLs e drivers suspeitos*

Essa etapa da metodologia tem o objetivo de aprofundar a análise dos possíveis códigos maliciosos identificados nas outras fases. Estes artefatos serão reconstruídos a partir do *dump* de memória que está sendo analisado. Os artefatos podem ser processos, *drivers* ou DLLs. O *hash* destes arquivos devem ser comparadas com uma base de *hash* de *malwares* conhecidos, como o Virustotal.

Deve-se procurar strings suspeitas, tais como url, email, CPF e nomes de arquivos de sistema. Além de identificar chamadas de sistemas comuns entre os *malwares* e a entropia dos arquivos.

H. *Processamento, Correlação e Relatório*

Os dados coletados e armazenados nas fases anteriores são correlacionados nesta fase e o relatório da análise é gerado. Para complementar as informações podem ser usadas diferentes fontes de *logs* sobre os possíveis *malwares*. Essas fontes podem ser histórico de navegadores, *eventlog*, *prefetch*, anomalias de *timeline* e data de compilação do arquivo executável.

I. *Criação da base de conhecimento*

Na última fase, deve-se utilizar padrões de descrição de indicativos de comprometimentos (IOC) para alimentar a base de conhecimento. Como exemplo, podem ser gerados IOC's baseado no *framework* OpenIOC (*framework open source* capaz de descrever as características comportamentais dos *malwares*[13]). No caso da Maldetect algumas verificações não estão descritas nestes padrões abertos e por isso será usado um padrão próprio para descrição dos IOC's encontrados.

VI. Técnicas de Automatização da Metodologia Maldetect para Windows

Foi construída uma ferramenta que implementa cada fase da metodologia Maldetect. A figura 5 mostra a tela inicial desta ferramenta, denominada de Maldetect Tool, a qual foi

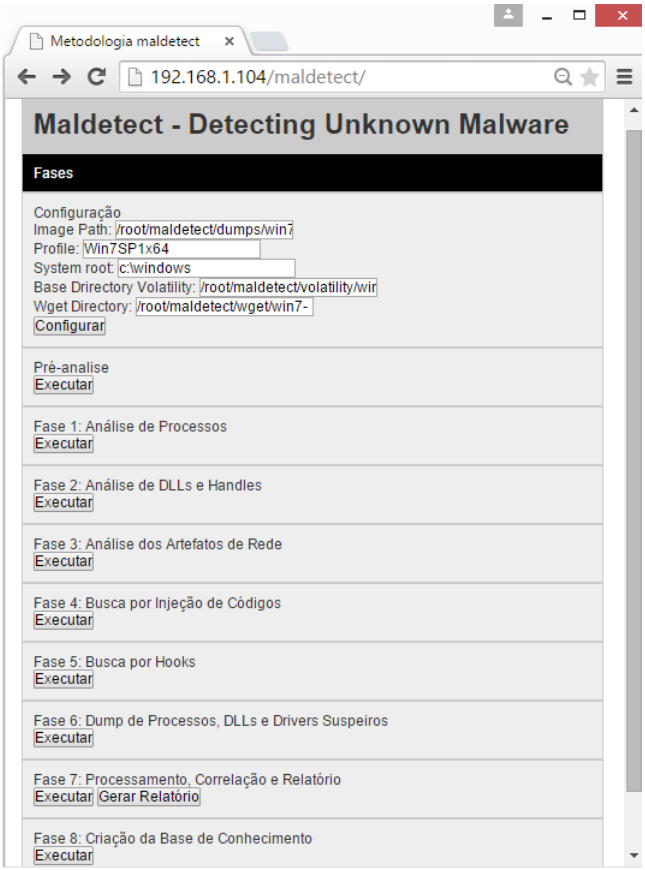

Fig. 5. Tela inicial da versão beta da *Maldetect Tool*.

implementa usando a linguagem PHP e Python. A figura 6 mostra a interação da Maldetect Tool com os recursos externos utilizados. Os principais recursos são: Volatility, VirusTotal, IPVoid[7] e a base de conhecimento. Foi criado um repositório no github (https://github.com/maldetect/maldetect) para disponibilizar os resultados das análises realizadas com a Maldetect Tool. A seguir serão descritas as técnicas utilizadas na construção da Maldetect Tool para o caso da análise de memória volátil do sistema operacional Windows 7.

A. *Pré-análise*

Esta fase recebe como entrada os IOC's gerados por execuções anteriores da metodologia maldetect. Estes arquivos serão criados no formato XML na última fase da metodologia. Dessa forma, é possível otimizar a detecção de códigos maliciosos existente na base de conhecimento. Além disso, deve ser coletada informações que auxiliarão a execução da metodologia, como por exemplo, portas de redes que devem ser consideradas como normais.

B. *Análise de processos*

Foi construído um *plugin*, denominado de procinfo, em Python para o Volatility para extrair as seguintes informações

[7]IPVoid é um serviço *online* e gratuito que faz análise de IP e DNS baseado em *blacklist*. Acessado em http://www.ipvoid.com/

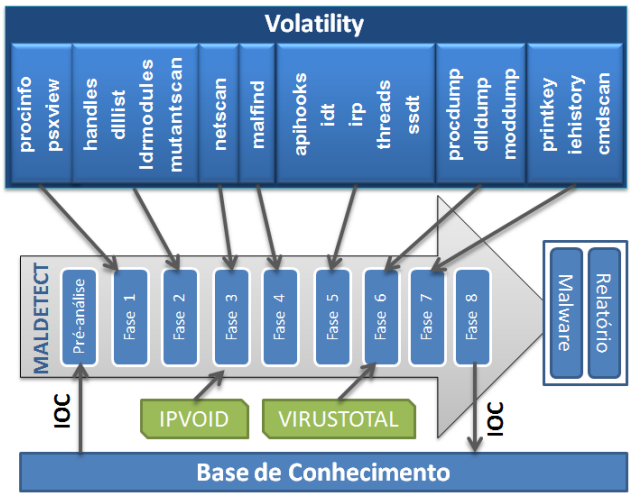

Fig. 6. Arquitetura da Maldetect Tool para o caso de análise de memória volátil do sistema operacional Windows 7. O *plugin* procinfo não faz parte da lista de *plugins* do Volatility e foi criado especialmente para a automatização da metodologia Maldetect.

do *dump* de memória a ser analisado: PID, PPID, PROCESS_NAME, BASEPRIORITY, PATH, COMMAND_LINE, SESSIONID, CREATE_TIME, EXIT_TIME, HANDLES, THREADS e USERNAME. A criação deste *plugin* foi necessária, pois os que já existem não mostram todas as informações consideradas relevantes para as verificações que devem ser realizadas nesta fase. O *process_analyser* realiza um *parser* das informações extraídas pelo procinfo e as carrega em um banco de dados MySQL. Nesta fase, os valores desses atributos são verificados se estão de acordo com a documentação da Microsoft para os seguintes processos do sistema operacional: System, Smss, Crss, Wininit, Services, Lsass, Svchost, Lsm, Winlogon, Explorer, Conhost, Rundll32, Taskhost e IExplore. Para cada um desse processos foi feita um pesquisa no banco de dados por processos com nomes com grau de similaridade maior que 80% com relação aos nomes dos processos do sistema operacional.

Para detectar processos ocultos (*hidden process*) foi utilizado o *plugin psxview* do Volatility, pois este acessa a lista de processos da estrutura _EPROCESS de maneiras diferentes tendo a capacidade de detectar um processo não vinculado.

C. Análise de DLLs e handles

Para verificar os *handles* do tipo *mutex*, muito usado pelos *malwares*, foi utilizado o *plugin mutantscan* do Volatility. Durante as análises pode ser criada uma *blacklist* de *mutex* maliciosos. Em [17], no ano 2014, foi apresentado que o *mutex* "2gvwnqjz1" é muito usado pelos *malwares*, pois em todos os casos onde foi encontrado estava associado a um código malicioso. A *blacklist* utilizada nesta implementação está disponível no repositório do github.

Para verificar quais DLLs não estão vinculadas foi utilizado o *plugin ldrmodules* do Volatility, o qual percorre a lista de DLL de formas diferentes. Foi usado o *plugin dlllist* do Volatility com o objetivo de listar as DLLs carregadas de cada processo e seu respectivo caminho no disco. Assim, foi possível realizar uma pesquisa no banco de dados para descobrir quais processos carregavam DLLs de conexão de rede (winsock32.dll, ws2_32.dll, wininet.dll e urlmon.dll), e com esse resultado é possível verificar e marcar quais desse processos tipicamente não fazem uso de conexões de rede, ou seja, averiguar possível discrepância entre as funcionalidades das DLLs e do processo que as carregou.

Nesta fase, foi feita a busca por características típicas de *backdoor*. Através do *plugin handles* do Volatility foi possível verificar se os processos cmd.exe possui um *handle* do tipo *File* com valor \Device\Afd\Endpoint, pois é um comportamento comum dos *Backdoors*[12].

D. Análise de Artefatos de Rede

Foi utilizado o *plugin netscan* do Volatility para listar as conexões de rede. A análise desta fase foi separada em duas etapas (portas em modo *Listening* e conexões estabelecidas). Para as portas em modo *Listening* foi feito um mapeamento das portas TCP que normalmente são abertas por uma máquina Windows 7 livre de *malwares*. Dessa forma, ao analisar o *dump* de memória em questão as portas que não estão nesta *whitelist* são marcadas como suspeitas. Nas portas com conexões estabelecidas, os IP's remotos foram submetidos ao IPVoid para verificação de sua reputação.

E. Busca por injeção de código

O *plugin malfind* do Volatility foi utilizado para mapear quais áreas de memória estão marcadas como PAGE_EXECUTE_READWRITE, pois essas são as possíveis áreas de injeção de código. Assim os processos associados a essas áreas são marcados como suspeitos.

F. Busca por hooks

Para cada técnica de *hook* utilizada pelos *rootkits* foi usado o *plugin* correspondente do Volatility. Para os hooks de IAT, EAT e *inline hook*, foi usado o *plugin apihooks*, e os outros os nomes dos *plugins* do Volatility correspondem a técnica de *hook* que eles detectam. Portanto foram usados os seguintes *plugins* do Volatility: *ssdt*, *irp*, *idt*. Além disso, foi feita a verificação de *threads* órfãs usando o *plugin threads* do Volatility.

G. Dump de processos, DLLs e drivers suspeitos

Nesta fase, são listados os processos em ordem decrescente do número de anomalias encontradas nas fases anteriores e o analista pode escolher os artefatos que serão extraídos da memória. Para realizar a extração destes artefatos foi utilizado os seguintes *plugins* do Volatility: *procdump*, *ddldump* e *moddump*. Após a extração, usando a API do VirusTotal é feita uma consulta pelo *hash* sha256 do artefato extraído do *dump* da memória.

H. *Processamento, Correlação e Relatório*

Todas as informações obtidas nas fases anteriores são processadas e correlacionadas de tal forma que os possíveis códigos maliciosos recebam uma pontuação para cada anomalia encontrada, quanto maior sua pontuação mais anomalias o processo possui. Para cada antivírus que retorna como *malware* a Maldetect Tool eleva a pontuação de anomalia do artefato. A descrição da cada anomalia é armazenada na base de dados para fins de relatório. Além disso, para compor o relatório final são coletadas as seguintes informações:

* valores da chave de registro Microsoft\Windows\CurrentVersion\Run (usada para armazenar programas que serão executados junto com a inicialização do windows);
* histórico de acesso do Internet Explorer;e
* histórico de comandos do cmd.exe.

Para capturar essas informações foram utilizados respectivamente os seguintes *plugins* do Volatility: *printkey*, *iehistory* e *cmdscan*. A Maldetect Tool gera um relatório em PDF contendo todos os artefatos que receberam uma pontuação de anomalias, assim como as anomalias de rede, o histórico de comandos do cmd.exe e acessos do Internet Explorer.

I. *Criação da base de conhecimento*

Na última fase da metodologia, o analista pode escolher quais processos ele gostaria de gerar o arquivo de IOC para compor a base de conhecimento. Este arquivo possui o formato XML onde são descritas as anomalias encontradas durante as fases anteriores da metodologia. Este arquivo irá alimentar a fase de pré-análise da próxima execução.

VII. Resultados e Discussões

Como prova de conceito da metodologia Maldetect e validação da implementação realizada pela ferramenta Maldetect Tool, foi feita uma análise automática de quatro *dumps* de memória de uma máquina Windows 7 infectada com os seguintes *malwares*: jackal[8], nfe.xml.exe[9], CiGPxdM.exe[10] e NF-e 18454310845.exe[11]. Foi analisado um *malware* menos conhecido pelos antivírus como é o caso do jackal.exe. Inclusive, antivírus como Kaspersky não o detectaram como sendo um código malicioso. A tabela I mostra a taxa de detecção desses *malwares* no VirusTotal.

TABELA I
Resultado da submissão dos códigos maliciosos ao VirusTotal.

Malwares	Taxa de detecção	Data da Análise
jackal.exe	20 / 57	26/04/2015 15:30:39 UTC
nfe.xml.exe	34 / 57	26/04/2015 15:34:43 UTC
NF-e 18454310845.exe	40 / 57	26/04/2015 15:40:05 UTC
CiGPxdM.exe	46 / 57	26/04/2015 15:37:12 UTC

[8]md5: e0208ab8930434036cbeef5683418d23
[9]md5: f6be0475e183335e00ffe363cf62a2bc
[10]md5: 116addcf779c596ad11a3fe910050c9e
[11]md5: f9856997401fd45a38790dcb1402537e

Foi feito um *dump* da memória da estação infectada com cada um dos códigos maliciosos. Os *dumps* de memória obtidos foram analisados automaticamente utilizando a Maldetect Tool. A ferramenta não sabia previamente nenhuma informação sobre o artefato malicioso que havia nos *dumps* de memória. A tabela II mostra os artefatos considerados maliciosos pela Maldetect Tool e as respectivas atividades típicas de *malwares* encontradas. Além dessas anomalias, o relatório também apresenta os registros do histórico do Internet Explorer, as anomalias de rede e o histórico de comandos dos processos cmd.exe que estavam em execução no momento da aquisição da memória. O relatório completo gerado para cada *malware* está disponível no repositório do Github[12].

TABELA II
Atividades típicas de códigos maliciosos encontrados e seus respectivos artefatos

Malwares	Artefato	Atividade maliciosa
jackal.exe	jackal.exe.exe	Cria dois processos cmd.exe! Mutex malicioso encontrado: __Dassara__! Porta ou conexão suspeita! Backdoor! Taxa de Detecção do Virustotal: 12 / 57! Carrega duas DLLs num contexto suspeito!
nfe.xml.exe	MALDETECT-PC.e	Este processo esta sendo executado a partir da pasta appData! Carrega uma DLL num contexto suspeito! Possui área de memória com a flag de write_exec!
CiGPxdM.exe	svchost.exe	Pai não encontrado! Caminho incorreto! Username incorreto! Falta parâmetro -k! Possui área de memória com a flag de write_exec!
NF-e 18454310845.exe	svchost.exe	Pai não encontrado! Caminho incorreto! Username incorreto! Falta parâmetro -k! Possui área de memória com a flag de write_exec!

Percebeu-se que os *malwares* tentaram dificultar sua detecção usando nomes de processos correspondentes à nomes de processos legítimos do sistema operacional, no caso foi usado o nome svchost.exe (nome de processo que pertence ao núcleo do sistema operacional Windows 7). O jackal foi executado a partir da pasta c:\windows\system32 na tentativa de se camuflar como um processo legítimo do Windows 7. Todos códigos maliciosos testados foram detectados e a automatização proposta, implementada pela Maldetect Tool, reduziu consideravelmente o tempo de análise da memória volátil em relação a análise manual. Por fim, o relatório gerado pela Maldetect Tool apresenta todas as informações relevantes coletadas durante a análise e direciona a atenção do analista para os artefatos que realmente realizam atividades

[12]https://github.com/maldetect/maldetect

típicas de *malware* deixando claro qual foi o artefato malicioso encontrado.

A Maldetect Tool ainda está em desenvolvimento, e está na versão *beta* para testes e melhorias das técnicas de detecção de anomalias comportamentais típicas de códigos maliciosos.

VIII. Conclusao

A metodologia *Maldetect* coleta e correlaciona informações comportamentais dos processos, DLLs e *drivers* de um *dump* de memória volátil (RAM) e identifica quais desses comportamentos são típicos de códigos maliciosos. A metodologia pode ser aplicada para detectar as ameças avançadas modernas e *malwares* desconhecidos. Além disso, é uma metodologia automatizável e independente de sistema operacional.

A metodologia *Maldetect* demonstra que a verificação de características comportamentais é mais eficaz que a detecção baseada em assinatura usada pela maioria dos antivírus. Utilizando a *Maldetect Tool*, que implementa a metodologia proposta, foi possível detectar os artefatos maliciosos baseado em características comportamentais. Além disso, a *Maldetect Tool* detectou *malwares* com baixa taxa de detecção no VirusTotal. Apesar do aumento da complexidade e do avanço das técnicas usados pelos *malwares* modernos, coletar e correlacionar informações comportamentais de várias fontes é uma das maneiras eficientes de detectá-los.

Em trabalhos futuros, sugere-se a ampliação da base de conhecimento dos indicativos de comprometimentos dos vários tipos de códigos malicioso com o objetivos de aplicar técnicas de aprendizado de máquina para verificar se ocorre agrupamento entre as características dos tipos de artefatos maliciosos e assim determinar qual o melhor tipo classificador.

Agradecimentos

Agradeço primeiramente a Deus, que me sustentou, capacitou e me deu saúde para realizar mais um projeto em minha vida. Aos professores, coordenadores, e funcionários do Departamento de Engenharia Elétrica da Universidade de Brasília que nos proporcionaram o ambiente saudável para pesquisa e desenvolvimento. Também ao orientador Dino, que sempre esteve disponível e paciente para me auxiliar nesta jornada. E por último, mas não menos importante, a minha esposa e família que me apoiaram e incentivaram nessa caminhada.

Referências

[1] Anand Ajjan. Ransomware: Next-generation fake antivirus. A SophosLabs technical paper, 2013. http://www.sophos.com/en-us/medialibrary/PDFs/technical%20papers/SophosRansomwareFakeAntivirus.pdf?la=en.pdf?dl=true.

[2] Michael Bailey, Jon Oberheide, Jon Andersen, Z Morley Mao, Farnam Jahanian, and Jose Nazario. Automated classification and analysis of internet malware. In *Recent advances in intrusion detection*, pages 178–197. Springer, 2007.

[3] Bill Blunden. *The Rootkit Arsenal: Escape and Evasion in the Dark Corners of the System*. Jones & Bartlett Publishers, 2011.

[4] DFRWS. The dfrws 2005 forensic challenge, 2005. http://www.dfrws.org/2005/challenge/index.shtml.

[5] Fahad Eshan. Memory forensics & security analytics: Detecting unknown malware, 2014. http://www.isaca.org/chapters5/Ireland/Documents/2014%20Event%20Presentations/Detecting%20Unknown%20Malware%20Memory%20Forensics%20and%20Security%20Analytics%20-%20Fahad%20Ehsan.pdf.

[6] Volatility Foundation. Command reference, 2015. https://github.com/volatilityfoundation/volatility/wiki/Command-Reference.

[7] Liang Hu, Shinan Song, Xiaolu Zhang, Zhenzhen Xie, Xiangyu Meng, and Kuo Zhao. Analyzing malware based on volatile memory. *Journal of Networks*, 8(11):2512–2519, 2013.

[8] Randall Hyde. *The art of assembly language*. No Starch Press, 2010.

[9] Rob Lee. Finding unknown malware – step-by-step. SANS DFIR Faculty, 2013. http://digital-forensics.sans.org/media/poster_fall_2013_forensics_final.pdf.

[10] Frankie Li. A detailed analysis of an advanced persistent threat malware. *SANS Institute InfoSec Reading Room*, 2011.

[11] Michael Ligh, Steven Adair, Blake Hartstein, and Matthew Richard. *Malware analyst's cookbook and DVD: tools and techniques for fighting malicious code*. Wiley Publishing, 2010.

[12] Michael Hale Ligh, Andrew Case, Jamie Levy, and Aaron Walters. *The Art of Memory Forensics: Detecting Malware and Threats in Windows, Linux, and Mac Memory*. John Wiley & Sons, 2014.

[13] Hun-Ya Lock. Using ioc (indicators of compromise) in malware forensics, 2013. http://www.sans.org/reading-room/whitepapers/forensics/ioc-indicators-compromise-malware-forensics-34200.

[14] Cameron H Malin, Eoghan Casey, and James M Aquilina. *Malware forensics: investigating and analyzing malicious code*. Syngress, 2008.

[15] Mihály Oroszlány. Rootkits under windows os and methods of their detection. Masaryk University Faculty of Informatics, 2008. http://is.muni.cz/th/139801/fi_b/Bc.pdf.

[16] Mark Russinovich, David A. Solomon, and Alex Ionescu. *Windows Internals, Part 1. 6th Edition*. Microsoft Press, 2012.

[17] Rob Seger. Hunting the mutex, 2014. http://researchcenter.paloaltonetworks.com/2014/08/hunting-mutex/.

[18] Johannes Stüttgen and Michael Cohen. Anti-forensic resilient memory acquisition. *Digital Investigation*, 10:S105–S115, 2013.

[19] Chad Tilbury. Memory forensics. SANS Computer Forensics and Incident Response, 2012. http://software.msu.montana.edu/free/ISSA/Memory%20Forensics%20Made%20Easy%20Solving%20Cases%20with%20the%20New%20Breed%20of%20Tools-Tilbury-5-22-2012.pdf.

[20] Stefan Vömel and Felix C Freiling. A survey of main memory acquisition and analysis techniques for the windows operating system. *Digital Investigation*, 8(1):3–22, 2011.

[21] Guangmingzi Yang, Zhihong Tian, and Wenliang Duan. The prevent of advanced persistent threat. *Journal of Chemical and Pharmaceutical Research*, 6(7):572–576, 2014.

[22] Boyun Zhang, Jianping Yin, and Jingbo Hao. Using fuzzy pattern recognition to detect unknown malicious executables code. In *Fuzzy Systems and Knowledge Discovery*, pages 629–634. Springer, 2005.