Universidade Católica Portuguesa
Faculdade de Engenharia

Segurança Informática - Vulnerabilidades Aplicacionais

Tiago Miguel Brito da Silva

Dissertação para obtenção do Grau de Mestre em Engenharia Informática

Júri

Prof. Doutor Rui Jorge Correia Mendes Alves Pires (Presidente)
Prof. Doutor Fernando Manuel Fernandes Melício
Prof. Doutor Tito Lívio dos Santos Silva (Orientador)

Outubro de 2012

Universidade Católica Portuguesa
Faculdade de Engenharia

Segurança Informática - Vulnerabilidades Aplicacionais

Tiago Miguel Brito da Silva

Dissertação para obtenção do Grau de Mestre em Engenharia Informática

Júri

Prof. Doutor Rui Jorge Correia Mendes Alves Pires (Presidente)
Prof. Doutor Fernando Manuel Fernandes Melício
Prof. Doutor Tito Lívio dos Santos Silva (Orientador)

Outubro de 2012

Resumo

Com o aparecimento dos computadores e com a atual dependência da população relativamente a estas máquinas, surge a necessidade de melhorar a segurança dos sistemas. Há alguns anos atrás que as vulnerabilidades por buffer overflow e DLL injection são consideradas críticas, pois são utilizadas muitas vezes para atacar computadores em rede. Um atacante anónimo, ao realizar um ataque deste tipo, procura obter acesso ilegal a um computador, muitos destes ataques derivam da aplicação de engenharia social. A aliança entre o acesso ilegal a um sistema por via de uma vulnerabilidade com a utilização da Engenharia Social, tem como objectivo a criação de um ataque híbrido. A fim de poder proteger um sistema, é necessário identificar as potenciais ameaças e por conseguinte conhecer e prever a maneira de proceder do possível atacante. Perante este cenário, foi necessário criar proteções que minimizem o risco de ataque muitas vezes partindo de formação dada aos utilizadores e estabelecer um conjunto de critérios que avaliem um sistema, quanto à sua confidencialidade, integridade e disponibilidade. Para provar que as novas funcionalidades de segurança dos atuais sistemas operativos são eficientes contra algumas destas vulnerabilidades, procurou-se neste trabalho construir um malware que testasse essas mesmas funcionalidades. Este programa engloba um ataque por via da injeção de uma DLL, seguido de um escalamento de privilégios, culminando com roubo de informação do utilizador. Na metodologia proposta procurou-se verificar em que condições os sistemas de proteção dos sistemas operativos cedem e permitem a instalação do malware. Através da metodologia proposta foi possível verificar os sistemas operativos que conseguem impedir o ataque. De modo a impedir e a melhorar a mais os sistemas operativos atuais é necessário novas formas de desenvolver software seguro, baseadas na aplicação das teorias existentes, como na adopção de um processo de desenvolvimento que considere os requisitos de segurança como parte integral do projeto de construção de software.

Este trabalho espera-se contribuir para a melhoria dos sistemas de segurança dos sistemas operativos.

Palavras Chave: Vulnerabilidades, Buffer Overflow, Engenharia Social, Segurança;

Abstract

With the advent of computers and the current dependence of population on these machines, there is a need to improve security systems. Since some years ago a buffer overflow and DLL injection vulnerabilities are considered critical, because they are often used to attack computers on the network. An anonymous attacker, when performing such an attack, seek illegal access to a computer, many of these attacks are provided from the application of social engineering. The alliance between illegal access to a system via a vulnerability with the use of Social Engineering, aims to create a hybrid attack. In order to protect a system, it is necessary to identify potential threats and therefore know and predict the attack possible way of act. Given this scenario, it was necessary to create safeguards that minimize the risk of attack, these measures may be the training given to users and establishing a set of criteria to evaluate a system, to its confidentiality, integrity and availability. To prove that the new security features of current operating systems are effective against some of these vulnerabilities, this study sought to build a malware that can test these same features. This program encompasses an attack by injecting a DLL, followed by an escalation of privileges, culminating with the theft of user information. In the proposed methodology attempts to verify the conditions under which protection systems operating systems give way and allow the installation of malware. Through the proposed methodology was able to verify the operating systems that can prevent the attack. In order to prevent and improve the most current operating systems need new ways to develop secure software, based on the application of existing theories, such as the adoption of a development process that considers the safety requirements as an integral part of the construction project software.

This work is expected to contribute to the improvement of safety systems operating systems.

Keywords: Vulnerability, Buffer-Overflow, Social Engineering, and Security;

Agradecimentos

A elaboração desta tese contou com a colaboração de pessoas, a que gostaria de salientar:

Ao Professor Doutor Tito Santos Silva agradeço a disponibilidade, a sabedoria e os ensinamentos constantes em todo o processo de orientação científica desta dissertação.

À minha família, em especial aos meus pais e irmã pelo apoio incondicional, compreensão nos momentos de maior indisponibilidade minha e por estarem sempre presentes.

Índice

Índice de Figuras

1. Introdução

2. Estado da Arte

3. Metodologia Proposta

Índice de Tabelas

1. Introdução

2. Estado da Arte

3. Metodologia Proposta

Lista de Algoritmos

1. Introdução

3. Metodologia Proposta

Lista de Abreviaturas

ASLR – Do inglês *Address space layout randomization,* que significa Aleatoriedade do Espaço de Dados;

BO – Do inglês *Buffer-Overflow,* que significa Estouro no Buffer;

CC – Do inglês *Commom Criteria;*

CPU – Do inglês *Central Processing Unit,* que significa Unidade Central de Processamento;

DEP – Do inglês *Data Execution Prevention,* que significa Prevenção de Execução de Dados;

DLL – Do inglês *Dynamic-Link Library,* que significa biblioteca de vínculo dinâmico;

OS – Do inglês *Operating System*, que significa Sistema Operativo;

PID – Do inglês *process identifier,* que significa identificador de processo;

RAM – Do inglês *Random Acess Memory,* que significa Memória de Acesso Aleatório;

SO –Sistema Operativo;

TCB – Do inglês *Trusted Computer Base,* que significa Base Confiável de Computação;

TCSEC – Do inglês *Trusted Computer System Evaluation Criteria,* que significa Critério de avaliação de confiança computacional;

VM – Do inglês *Virtual Memory,* que significa Memória Virtual;

IE – Internet Explorer;

SACL – Do inglês *System access control lists* **,** que significa Listas de controlo de acesso ao sistema;

DACLs – Do inglês *Discretionary access control lists*, que significa Lista de controlo de acesso discricionário.

1. Introdução

A questão da segurança nos Sistemas de Informação, amplamente discutida pelos meios de comunicação atuais, torna-se um grande problema quando o assunto é a escolha das ferramentas, metodologias para a sua prevenção e combate. Independentemente dos métodos utilizados em diversas empresas tenham ou não sucesso, a questão é mais complicada quando o problema advém de aspectos humanos. Estes não têm recebido a devida atenção quando se procura uma solução para alguns problemas de segurança. Hoje em dia, a informação é o ativo mais valioso numa organização. Ao mesmo tempo passa a ser também a mais visada e ambicionada por pessoas mal intencionadas com o objetivo de investigar por curiosidade, furtar para obter informações sigilosas e valiosas, criar danos por diversão, por beneficio próprio ou mesmo até por vingança. Por isso existe uma enorme preocupação em relação à segurança da informação nas organizações ou até mesmo em lares particulares, pois representa muitas vezes roubo de ideias ou bens, e obtenção de lucro através disso.

O problema é que as organizações dão apenas importância à atualização dos seus sistemas tecnológicos, implementando tecnologia de última geração, que muitas vezes podem não ser livres de vulnerabilidades internas. Claro que toda a tecnologia é importante e fundamental para a segurança da informação, mas não é tudo. De nada vale proteger um sistema contra invasões se for alguém de dentro da empresa a dar o acesso ao atacante, dessa maneira todo o investimento foi desnecessário.

Infelizmente ainda não é muito habitual as empresas investirem na formação dos seus funcionários, pois estes também fazem parte da segurança da informação da empresa. Quanto mais evoluem os sistemas e os dispositivos de segurança mais os invasores irão explorar a vulnerabilidade humana. Perante as vulnerabilidades descobertas nos sistemas, os invasores farão uso de técnicas que ludibriem os colaboradores de uma empresa ou um simples utilizador comum. Hoje em dia é bastante comum fazer-se uma aliança entre vulnerabilidades técnicas e a ignorância dos utilizadores desses mesmos sistemas, pois a união permite um maior aproveitamento da vulnerabilidade.

1.1. A Vertente da Engenharia Social

O termo "engenharia social" (do inglês "social engineering"), representa a arte de influenciar pessoas a fim de controlar mecanismos de segurança. Esta técnica compreende em obter informações, por parte dos utilizadores por correio electrónico, por telefone, ou por contacto direto.

A engenharia social tem como base a utilização da força de persuasão e na exploração da inocência dos utilizadores, fazendo-se o atacante frequentemente passar por uma pessoa que não é, ou consistindo o ataque de um programa com comportamento falso. Os objetivos básicos de engenharia social são o mesmo que hacking em geral, para obter acesso não autorizado a sistemas ou informações, a fim de cometer fraude, invasão de rede, espionagem industrial, roubo de identidade, ou simplesmente para perturbar o sistema ou rede.

Geralmente, os métodos de engenharia social desdobram-se de acordo com o seguinte esquema:

- Uma fase de abordagem que permite colocar o utilizador à vontade fazendo-se passar por uma pessoa da sua categoria , do seu meio ou da sua empresa;

- Um alerta, com o objectivo de destabilizar e assegurar-se da rapidez da sua reação. Pode tratar-se, por exemplo, de um pretexto de segurança ou de uma situação de imprevisto; e

- Uma fase final em que tudo corre como o utilizador esperava, de modo a deixar o utilizador tranquilo e presumindo que nada de anormal ocorreu.

A figura abaixo ilustra o ciclo de ataque da engenharia social este ciclo consiste em quatro fases:

- reunir informações;
- desenvolver o relacionamento com a vitima;
- exploração; e
- execução.

Cada ataque de engenharia social é único, com a possibilidade de envolver múltiplas fases e pode mesmo agregar o uso de outras técnicas de ataque mais tradicionais para atingir o resultado desejado (Malcolm A, 2007)

Fig. 1.1 - Ciclo de um ataque de engenharia social

Encontrar bons exemplos na vida real de ataques de engenharia social é difícil. As organizações-alvo por vezes não querem admitir que foram vítimas (afinal, admitir uma violação de segurança fundamental é não só constrangedor, pode prejudicar a reputação da organização), e por outras vezes o ataque não foi suficientemente documentado para que alguém realmente tenha certeza se houve um ataque de engenharia social ou não.

Quanto ao porquê de as organizações serem atacadas através de engenharia social, é muitas vezes uma maneira mais fácil de obter acesso ilícito do que a maioria das formas de pirataria técnica. Ataques de engenharia social têm lugar em dois níveis: o físico e o psicológico. Primeiro, o ambiente físico para esses ataques: o local de trabalho, o telefone, o lixo, e até mesmo on-line. No local de trabalho, o hacker pode simplesmente enganar os funcionários da organização quanto à sua identidade, usando ideias pré-concebidas dos mesmos para evitar a sua própria identificação.

A Internet é um terreno fértil para os engenheiros sociais que procuram senhas. A principal fraqueza é que muitos utilizadores frequentemente repetem o uso de uma senha simples em cada conta: Hotmail, gmail, qualquer que seja. Assim, uma vez que o hacker tem uma senha, este provavelmente pode entrar em várias contas. Um dos meios mais utilizado pelos hackers para obterem esse tipo de senha é através de um formulário on-line: podem enviar algum tipo de informação sobre sorteios e pedir que o utilizador coloque um nome (incluindo e-mail) – deste modo o atacante pode até obter a senha da conta da pessoa.

Outra maneira dos hackers poderem obter informações on-line é, fingirem ser o administrador da rede, enviando e-mails através da rede e pedindo a senha do utilizador. Muitas das vezes a restrição dos acessos não se encontra bem definida e por essa razão, os hackers podem-se fazer passar por outras pessoas.

Por exemplo, anexos de email enviados por alguém podem levar vírus, worms e cavalos de tróia. Um bom exemplo disso foi um AOL hack, documentado pela VIGILANTe (VIGILANTe "Social Engineering", 2001). "Nesse caso, o hacker fazendo-se passar por um técnico de suporte da AOL, falou com utilizador que pediu apoio durante uma hora. Durante a conversa, o hacker mencionou que o seu carro estava à venda mais barato que o preço praticado em stands de venda em segunda mão e a pessoa que precisava de ajuda mostrou-se interessada, então o hacker enviou um anexo de e-mail "com uma imagem do carro ". Em vez de uma foto do carro, o atacante anexou um executável que realizou um exploit backdoor abrindo uma conexão fora do AOL através da firewall.

Os hackers de engenharia social conseguem a partir de um ponto de vista psicológico, criar um ambiente perfeito para o ataque. Métodos básicos de persuasão incluem: a representação, integração, conformidade, difusão de responsabilidade e simpatia. Independentemente do método utilizado, o objetivo principal é convencer a pessoa a revelar a informação ao engenheiro social.

A outra chave importante é nunca pedir muita informação de cada vez, mas pedir um pouco a cada pessoa, a fim de manter a aparência de uma relação confortável.

A *personificação* geralmente significa a criação de algum tipo de personagem e interpretação de um papel. Quanto mais simples o papel a interpretar, melhor, pois o hacker com um papel simples consegue ser mais credível. Muitas vezes, os hackers estudam um indivíduo real numa uma organização e esperaram até que a pessoa observada esteja fora da cidade para personificá-lo por telefone.

Alguns perfis que podem ser utilizados em ataques de representação incluem: um técnico de suporte de TI, um gerente, ou um colega de trabalho. Numa grande empresa, isso não é tão difícil de fazer, pois existem muitas pessoas e estas não se conhecem todas. A maioria desses papéis enquadram-se na categoria de alguém com autoridade. As vitimas não questionam uma pessoa que demonstre alguma autoridade, pois como não conhecem todos os colaboradores da empresa, iram divulgar informação

A conformidade é um comportamento baseado em grupo, mas pode ser usado ocasionalmente no campo individual, convencendo o utilizador de que todos deram a informação ao atacante. Quando o atacante atacar e for feita uma investigação, o funcionário estará seguro pois pensa que todos estão implicados como ele.

Área de Risco	Táctica do Atacante	Estratégia de Prevenção
HelpDesk	Persuasão e Personificação	Não divulgar passwords ou outra informação confidencial pelo telefone
Entrada em instalações	Acesso físico não autorizado	Vigilância de identificação
Escritório	Olhar sobre o ombro de um colaborador	Não introduzir passwords com alguém ao lado
Escritório	Circular à procura de secretárias vazias	Acompanhar visitas diretamente à saída
Sala de Correio	Inserção de avisos falsos	Trancar e vigiar sala de Correio
Sala das Máquinas	Tentativa de ganhar acesso para remover equipamento ou roubar dados confidenciais	Trancar sala dos telefones, o Datacenter e manter um inventário atualizado do equipamento
Telefone	Roubar acessos telefónicos	Controlar chamadas de longa distancia e rastrear chamadas
Lixo	Remexer o lixo	Manter o lixo em áreas seguras e vigiadas, destruir dados confidenciais
Intranet-Internet	Criação e inserção de software de roubo e rastreio de password na rede	Mudanças contínuas na sensibilização da rede
Escritório	Roubo de informação sensível	Marcar documentação como confidencial e trancar os mesmos documentos
Geral	Personificação e persuasão	Manter os funcionários informados e com formações sobre o problema

Tabela 1.1 – Tácticas utilizadas em Engenharia Social

A melhor maneira de obter informações num ataque de engenharia social é apenas ser agradável. O atacante deve ser aliciante de modo a conseguir ganhar a confiança do funcionário, ou seja, tem de ser credível.
Além disso, a maioria dos funcionários responde e o hacker sabe quando parar de fazer questões sobre a informação que necessita, pouco antes de o funcionário suspeitar que alguma coisa está errada.

Alguns pontos fundamentais na proteção contra ataques de engenharia social:

- Limitar o número de contas de utilizadores com privilégios na organização e o nível de acesso que eles têm, isso irá ajudar a limitar o dano que um ataque de engenharia social bem-sucedido possa causar;

- Regularmente rever as contas dos utilizadores. Fornecer acessos apenas para os que devem ter acesso e os recursos específicos para quem realmente precisa; e

- Verificar se as contas de utilizador têm uma autenticação forte.

A melhor arma contra a engenharia social é a informação, de nada adianta as empresas usarem sistemas muito protegidos se os seus funcionários não tiverem cientes dos ataques que podem sofrer.

De modo a compreender como são realizados ataques híbridos de engenharia social e o acesso ilícito a um sistema é necessário compreender o que é um sistema operativo e como decorreu a sua evolução ao longo dos anos.

1.2. A Vertente Técnica

1.2.1 Perspectiva Histórica

Nos anos 50 os primeiros sistemas informáticos não dispunham de software, logo não havia necessidade de um Sistema Operativo. Nesse tempo o objectivo era a implementação de hardware, e aperfeiçoar pouco a pouco fiabilidade e melhorar o desempenho. Só com o aumento da fiabilidade é que se começou a optimizar a utilização da máquina de modo a rentabilizar o hardware (Marques, 2009).

O Monitor de controlo foi a primeira grande prioridade, pois era um programa utilitário que possibilitava ao utilizador carregar os seus programas em memória, editá-los e verificar a sua execução. A gestão do sistema era bastante acessível, eram atribuídas aos utilizadores sessões de tempo de utilização, e durante esse tempo o utilizador dispunha do sistema na totalidade para realizar as operações necessárias, através dos comandos do Monitor. No fim os resultados eram guardados em folhas de papel perfuradas ou em fitas magnéticas.

Com o passar dos anos, a evolução da tecnologia e as necessidades que a própria evolução exigia, os sistemas operativos possibilitaram a multi-programação, memória virtual, timesharing e real time. Finalmente, o sistema operativo assemelha-se a uma máquina virtual que disfarça todos os detalhes físicos da máquina física num conceito que virtualiza o hardware e os mecanismos de baixo nível. Os sistemas passaram a ser multi-programados, o que leva à sobreposição entre as aplicações a um nível mais elevado, em que as aplicações competem pela utilização do CPU. Estes sistemas evoluíram para um sistema de gestão de memória, que foi melhorando ao longo do tempo.

Ao ser possível utilizar Memória Virtual num SO, é possível o programador ter uma abstração do espaço de endereçamento virtual de grande dimensão e independente da memória física livre.
Com o avançar destas características nos Sistemas Operativos, era necessário uma evolução nos microprocessadores, ou seja os dois dependem entre si para evoluir. Os microprocessadores foram melhorando com a tecnologia, hoje procura-se construir microprocessadores com uma capacidade de processamento elevada, mas com tamanho reduzido (ordem dos manómetros) (A evolução dos microprocessadores, 2012).

Com o aparecimento dos sistemas operativos e com a dependência social atual sobre estes, a segurança nos SO teve que ser melhorada. Com a utilização dos computadores como ferramentas de trabalho e de lazer, as falhas de segurança passaram a implicar a possibilidade de grandes prejuízos. A segurança no Sistema Operativo previne os atacantes de atingir os seus objectivos através do acesso e do uso não autorizado de computadores ou redes de computadores. Os Sistemas Operativos devem ser seguros, ou seja, devem fornecer informações intactas apenas a utilizadores autenticados e autorizados, somente quando as informações são pedidas através de requisições validas e identificadas. As informações não podem ser recebidas, modificadas, observadas por terceiros não autorizados.

1.2.2 Arquitetura de memória

Um processo em execução usa memória, mas a memória física é um recurso escasso e a gestão do endereçamento de programas é uma tarefa complexa. Uma das funcionalidades que o sistema operativo realiza é a gestão de espaços de endereçamento dos processos, através de uma apropriada gestão da memória principal (Tanenbaum, 2001).

A memória principal é designada por RAM – Random Access Memory e a memória secundária é geralmente constituída por discos magnéticos. A gestão da hierarquia de memória admite a criação de uma memória virtual, que permite uma dimensão bastante superior à memória física, explorando o melhor rácio entre a velocidade de acesso, o custo da RAM e da memória secundária. Perante esta capacidade é possível simular uma memória mais ampla minimizando a redução de desempenho. A parte do Sistema Operativo que é responsável pela gestão da hierarquia da memória é o Memory Manager.

A sua função é manter o controlo entre as partes da memória que estão em uso e as que não estão a ser utilizadas, alocar memória para computorizar processos, e como libertar memória quando o processo estiver terminado. Outra função é organizar a troca entre a memória principal e a memória secundária quando a principal é reduzida demais para correr os processos.

1.2.3. Gestão de Memória

Num sistema operativo o módulo de gestão de memória efetua a gestão e a optimização da memória física, admite a memória virtual dos processos e executa um conjunto de algoritmos associados à manipulação do espaço de endereçamento dos processos.
Este módulo é responsável sobretudo pela transferência da informação entre a memória física e secundária.

O módulo de gestão de memória pode ser considerado segundo dois aspetos: mecanismos de gestão de memória e algoritmos de gestão de memória. Estes algoritmos definem as decisões que devem ser tomadas e quando devem ser tomadas, usando mecanismos de baixo nível para as levar a cabo (Tanenbaum, 2001).

Os mecanismos de gestão de memória delimitam a organização da memória do computador, isto é, se o endereçamento é real ou virtual, se a memória é segmentada ou paginada e quais os seus tamanhos respectivamente.

Os endereçamentos de memória modificam-se durante a execução dos programas, a pilha altera-se dinamicamente e a zona dos dados pode ser modificada através da criação dinâmica de estrutura de dados. O isolamento desses espaços de memória é algo que deve ser controlado e garantido pelo SO.

O espaço de endereçamento de memória de um processo é o conjunto de posições de memória que um programa executado por esse processo pode referenciar. O espaço de endereçamento tem uma disposição que é estabelecida pelo sistema operativo, definindo as gamas de endereços usadas para o processo em modo utilizador. Nos sistemas Linux e Windows o espaço de endereçamento está dividido em duas grandes secções: a secção de nível de sistema e a de nível utilizador .

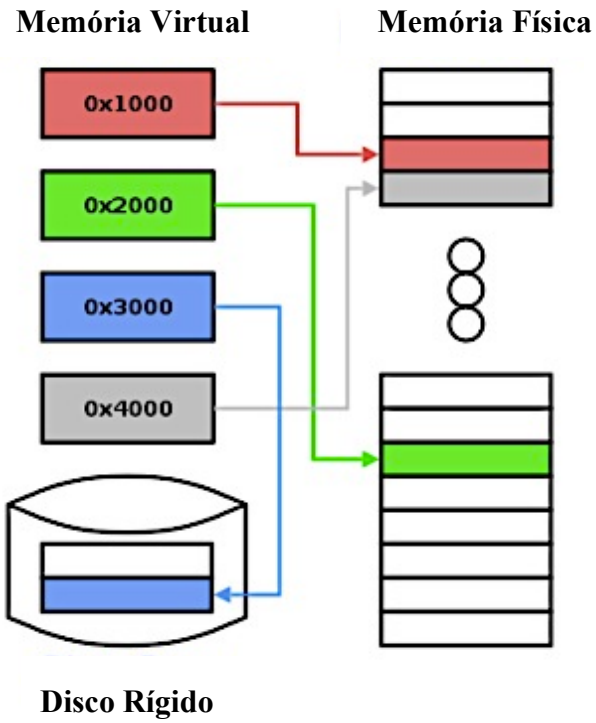


Fig. 1.2 - Relação entre a Memória Virtual e a Memória Física

A dimensão e a forma de gerir o espaço de endereçamento são totalmente dominadas pela tecnologia utilizada para a gestão da memória. Existe uma diferença entre a utilização de um espaço de endereçamento relacionado com a memória física e um espaço de endereçamento virtual que não sofre das restrições da memória física. Um espaço de memória virtual torna a tarefa do programador mais simples pois permite que este ignore o funcionamento da memória do processo, delegando essa função para o Sistema Operativo.

Se uma posição de memória for referenciada por um programa fora do espaço de endereçamento do processo que o suporta, o hardware de gestão de memória desencadeia uma exceção que será tratada pelo SO, caso não seja possível solucioná-la o programa é terminado. Existe um clara noção de confinamento do processo ao seu espaço de endereçamento válido, isto é, mesmo que teoricamente o processo possa endereçar toda a memória disponível, o SO tem em cada instante, um mapa preciso de quais as posições a que o programa pode aceder e de que forma o pode fazer. Este confinamento é assegurado pelos mecanismos de proteção de memória.

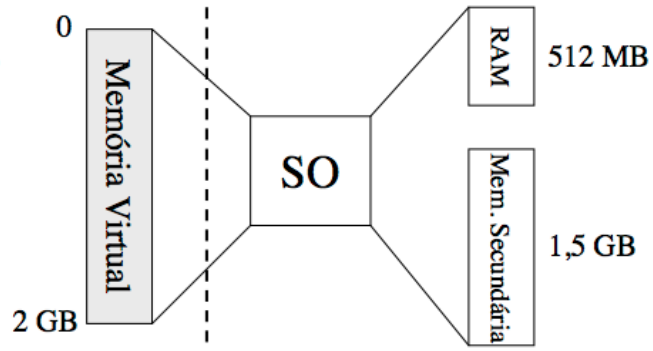


Fig. 1.3 - Memória Virtual

Memória Virtual

Há alguns anos atrás fomos confrontados com programas com dimensões grandes demais para caberem na memória disponível. Uma solução encontrada para este problema foi a Memória Virtual (MV). A ideia principal que se encontra sobre a MV é que o tamanho combinado entre o programa, os dados e a pilha podem exceder a quantidade de memória física disponível. O sistema operativo mantém algumas partes do programa na memória física e o resto no disco.

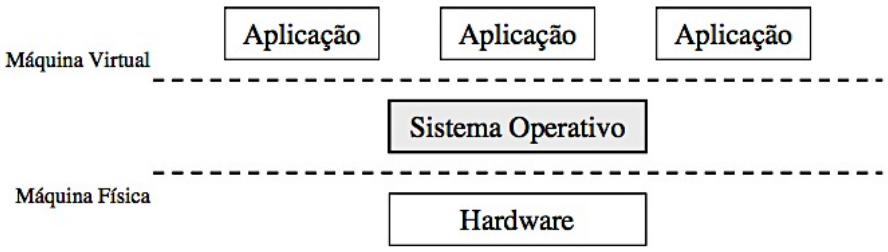


Fig. 1.4 - Localização do Sistema Operativo

Memória Partilhada

Como o nome indica, a memória partilhada cria um segmento de memória acessível para mais de um processo. Chamadas especiais ao sistema, ou pedidos para o kernel, podem alocar, libertar a memória e definir permissões assim como ler e escrever permitem colocar e obter os dados dessa região.

A memória partilhada não é desenhada a partir da memória própria de um processo, pois essa memória é sempre privada. Em vez disso, a memória partilhada é alocada na memória livre da pool do sistema e está anexada a cada processo que lhe quer ter acesso. Anexação é chamada de mapeamento, onde ao segmento partilhado de memória é atribuído endereços locais no espaço de cada processo.

Assumindo que dois processos, A e B, são executados no mesmo sistema, como mostrado na Figura 1.5, e foram especificamente preparados para coordenar e partilhar a informação através de memória partilhada. A e B têm tamanhos desproporcionais na figura para representar que as aplicações não precisam ser idênticas.

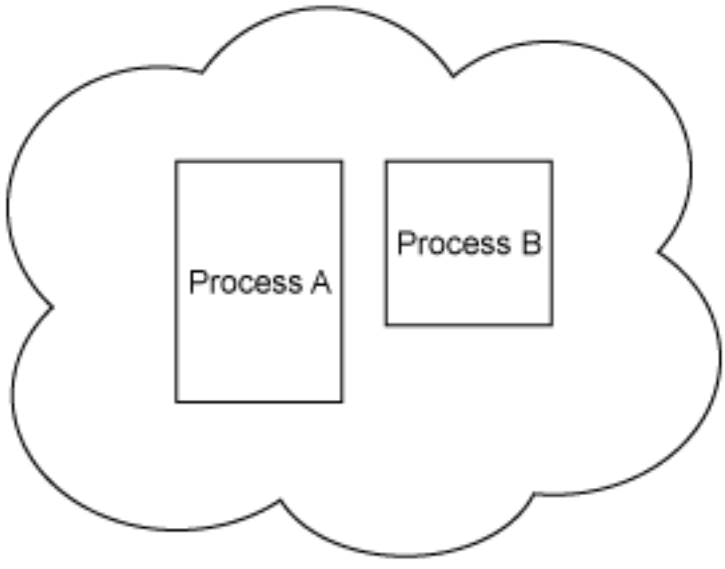


Fig. 1.5 - Dois processos a correr no mesmo sistema, executando código diferente

Na Figura 1.6, o processo A pede um segmento de memória partilhada, nesse instante o processo inicializa esse segmento de memória, preparando-o para uso. O processo nomeia o segmento para que outros processos possam encontrá-lo. Tipicamente, o nome do segmento não é atribuído dinamicamente, em vez disso, é bem conhecido e facilmente referenciado de outro código.

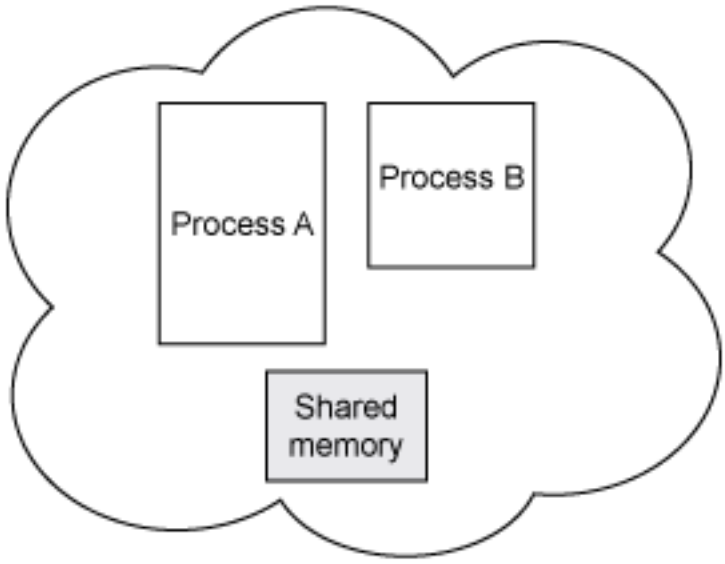


Fig. 1.6 - Um processo pedindo um segmento de memória partilhada

O processo A mapeia ou anexa o segmento de memória partilhada ao seu próprio espaço de endereços. Por outro lado, o processo B encontra o segmento de memória, através do seu pipe e nesse momento mapeia ou anexa, o segmento no seu espaço de endereços. Isto é mostrado na Figura 1.7.

Ambos os processos são ampliadas pelo tamanho do segmento de memória partilhada, que ambos partilham.

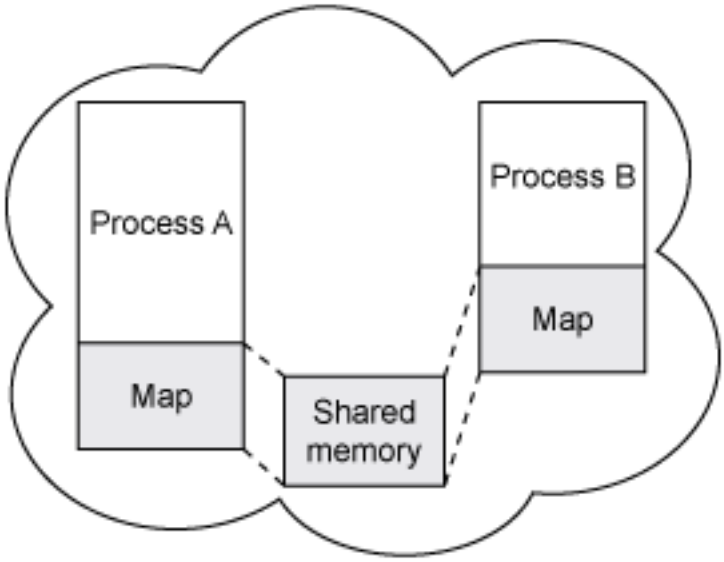


Fig. 1.7 - Ambos processos anexam, ou mapeiam, o segmento de memória partilhada

Finalmente, na Figura 1.8 o processo A e B podem ler e escrever a partir do segmento de memória partilhada livremente. A memória partilhada é tratada como a memória do processo local, ou seja propriedades de read () e write () funcionam normalmente.

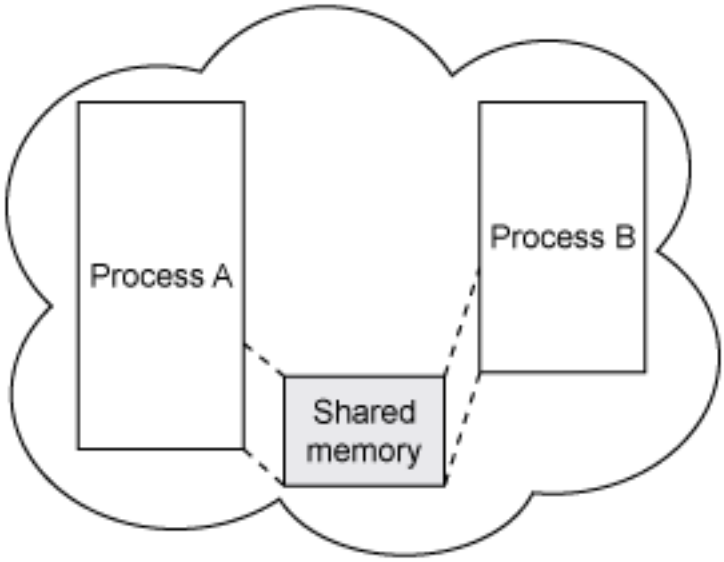


Fig. 1.8 - Dois ou mais processos podem partilhar dados via memória comum

1.2.4 Proteção de recursos

O sistema operativo tem como uma das suas principais funções, realizar a gestão dos diferentes recursos do computador. Esses recursos são geralmente designados por objetos: memória, discos, redes, impressoras, ficheiros, bibliotecas, entre outros.

As aplicações usam hardware (recursos) para atingir objectivos tais como ler e guardar dados, editar documentos, navegar na internet, reproduzir música/filmes, etc (Maziero. 2011). O SO deve garantir o uso apropriado destes recursos, perante os utilizadores que são adiante designados por sujeitos (Correia, 2010).

Sob o ponto de vista da segurança, o sistema operativo deve impedir que um utilizador aceda a um objecto de modo não autorizado.

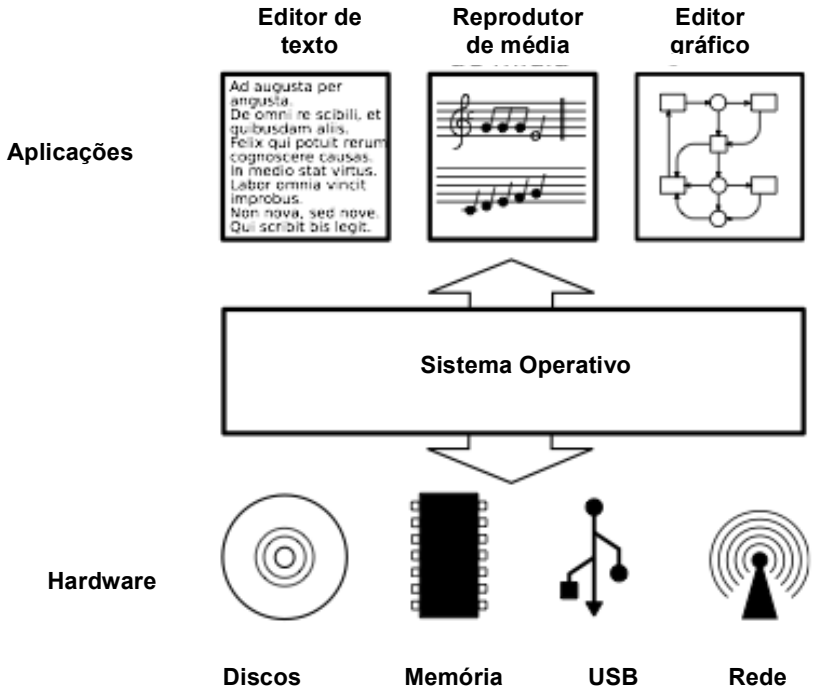


Fig. 1.9 – Estrutura de um sistema computacional típico.

1.2.5. Funcionamento do processador

O acesso arbitrário à memória deve ser impedido, para isso é necessário separar os utilizadores dos objetos. Essa separação é assegurada pelo sistema operativo e pelo processador. Nas arquiteturas convencionais o processador funciona em dois modos:

- **Modo Kernel -** o modo Kernel é o modo como corre o núcleo do sistema operativo, é onde ocorre o escalonamento dos processos, gestão de memória, etc. Neste modo não há restrições de acesso entre os recursos do sistema e o próprio software do SO.

- **Modo Utilizador –** o modo utilizador é modo em que o software que não pertence ao núcleo corre. Este modo restringe o acesso arbitrário entre o software e os recursos do sistema. Esta limitação é executada impedindo o software de correr certas instruções no processador. No modo utilizador, a execução de uma instrução de alto nível gera uma exceção.

Esta divisão em dois modos, ou seja a dois níveis básicos de privilégios, é a mais consensual hoje em dia (Correia, 2010).

Coloca-se uma questão face a estes dois modos, apesar correrem em modo utilizador, os processos precisam de executar operações de alto nível como por exemplo, ler do teclado, escrever no monitor, etc.
A solução para esta questão consiste em chamar um procedimento do núcleo que realiza a função pretendida, mas perante esta solução surgem duas dificuldades:

- É necessário o processador mudar de modo, pois o código do núcleo corre num modo diferente e mais privilegiado.

- A memória usada pelo núcleo é oculta para o processo em causa, logo, para o processo que realiza o pedido ao sistema, não existem endereços de procedimentos do núcleo para onde este possa deslocar-se.

Para deslocar o controlo entre os dois modos de execução, é necessário utilizar uma suspensão de software. Uma suspensão de software força o processador mudar para o modo Kernel e a executar a rotina de processamento de suspensão. Após essa suspensão, a rotina vai depois executar o processo indicado pelo programa.

A divisão do processador em dois modos de atividade representa um papel primordial na segurança dos computadores atuais. Este mecanismo chegou ao estado atual devido à necessidade de solidez e não tanto de segurança.

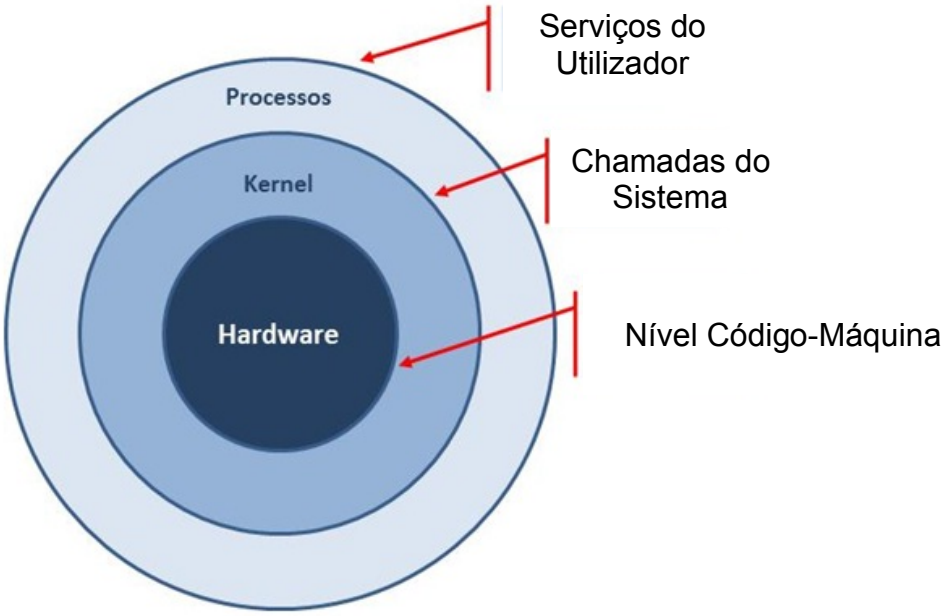


Fig. 1.10 – Distribuição do Microprocessador

1.2.6. Tipos de ataque à memória

Existem diversos tipos de ataque à memória de um Sistema Operativo, ataques esses que visam muitas vezes vulnerabilizar sistemas. Nem sempre estes ataques implicam más intenções por parte do atacante, mas muitas das vezes os ataques podem ser realizados com o objectivo de vandalizar um sistema e outras vezes para roubar ou alterar informação.

Por esta razão as ameaças à segurança de um sistema podem ser classificadas como acidentais, intencionais, ativas e passivas:

- **Ameaça Acidental:** existe sem intenção premeditada. Um exemplo é um mau funcionamento do sistema.

- **Ameaça Intencional:** pode ir desde uma "vista de olhos" pelos dados e sistema até sofisticados ataques usando profundos conhecimentos do sistema operativo.

- **Ameaça Passiva:** é aquela que, quando realizada, não resulta em nenhuma modificação da informação do sistema ou do estado do sistema.

- **Ameaça Ativa:** alteração da informação ou do estado do sistema.

Alguns dos ataques mais comuns são os de buffer-overflow, corridas e DLL Injection. De entre estes três ataques, o buffer-overflow permite despoletar os outros dois. Por esse motivo, é um dos ataques mais perigosos.

1.2.7. Ataque Buffer-OverFlow

Os buffer-overflows são a ameaça de segurança mais comum em sistemas de software hoje em dia e muitas da vulnerabilidades existentes devem-se a buffer-overflows (B.O).
Qualquer mitigação desta vulnerabilidade iria ter um grande impacto na melhoria da segurança dos nossos computadores.

Num ataque de B.O. num programa vulnerável, o atacante tenta modificar o estado da memória do programa para que esta permita que o atacante controle a máquina, preferencialmente no modo administrativo (com privilégios). Para iniciar um ataque o atacante tem de enviar cuidadosamente um excesso de dados de entrada para o programa. Um programa que não faça uma verificação do tamanho dos dados de entrada irá copiar os dados num local contíguo ao buffer.
Por meio de controlo do conteúdo o atacante pode fazer com que o programa desvie da sua finalidade (Olatunjz, 2003). Um exemplo clássico deste tipo de ataque é o da stack smashing, (Aleph, 1994) que simplesmente substitui o endereço de retorno de uma função na pilha de modo a que, quando a função retornar irá saltar para um local onde o atacante colocou código malicioso.

Detectores de buffer overflow:

Detectores Estáticos:

Os detectores de buffer overflow estáticos tentam verificar se todos os acessos à memória sofrem de overflow (Dor, 2003). Ferramentas inadequadas e mal configuradas, levam a perca de erros no código. Além disso os avisos gerados pelas ferramentas de análise exigem que o programador inspecione o código manualmente. Os detectores estáticos de buffer overflow devido a estes detalhes tornam-se impraticáveis de utilizar.

Detectores Dinâmicos:

Estes detectores são interessantes pois inserem automaticamente as proteções necessárias. Mas para um detector dinâmico ser implementado é fundamental que a proteção de buffer overflow, não quebre código de trabalho, ou seja dever ser feita automaticamente, sem intervenção do utilizador e seja razoavelmente eficaz.

Apesar dos esforços anteriores em auditoria de software, os buffers overflow continuam a ser descobertos em programas de uso comum. Um verificador de buffer overflow, não é nada mais que um limitador dinâmico de barreiras, que detecta abusos ao espaço disponível no buffer antes de estes ocorrerem, impedindo assim o ataque à integridade do sistema.

1.2.7.1. Buffer-OverFlow na Pilha

Alguns ataques devem-se ao facto de muitos sistemas operativos e outros programas de sistema serem escritos na linguagem de programação C. Infelizmente o compilador de C não faz verificação dos limites dos arrays. (Tanenbaum, 2001)

O resultado é que algum byte fora do array pode ser reescrito, o que levará a consequências desastrosas. Nenhuma verificação é efectuada em tempo de execução que verifique e previna este erro.

Na figura (a) seguinte, vê-se um programa main a correr, com as suas variáveis locais na pilha. Num determinado momento é chamado um procedimento A, mostrado na figura (b), a chamada acaba por re-escrever o endereço de retorno (que aponta para instruções seguintes à chamada) na pilha. Então transfere o controlo para A, que decrementa a o apontador da pilha para alocar espaço para as suas variáveis locais.

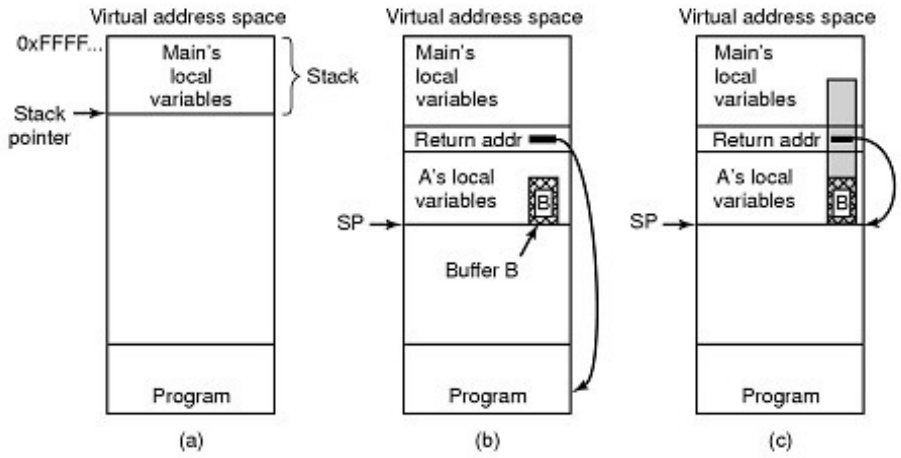


Fig. 1.11 - Ataque de Buffer-Overflow na Pilha

Admita-se que A necessita adquirir um path completo (concatenando a diretoria com o ficheiro) para o abrir. A tem um buffer de tamanho definido (array) B para receber o nome de um ficheiro (b). Usar um tamanho fixo do buffer para receber o nome do ficheiro é muito

mais fácil de programar do que calcular o tamanho e dinamizar a alocação de espaço necessário. Se o buffer for de 1024 bytes e admitindo que o sistema operativo limita nomes de ficheiros a 255 caracteres é presumível que não haja problemas. Este raciocínio tem uma falha, porque se o sistema for receber um nome com 2000 caracteres, não irá conseguir abrir o ficheiro (o que para o atacante é indiferente).

Mas quando o processo copiar para o buffer o nome do ficheiro irá reescrever sobre a memória como apresentado na figura (c). Pior ainda é se o nome for comprido demais irá reescrever sobre o endereço de retorno, quando A retornar irá retornar para um endereço que esteja no meio do nome do ficheiro. Se esse endereço for lixo, fará com que o programa salte entre endereços aleatórios e provavelmente irá finalizar de forma inesperada.

1.2.7.2. Exemplo de Buffer-Overflow na Pilha

Para melhor compreender uma vulnerabilidade deste tipo, é necessário observar um pouco de código C, que esteja protegido e não protegido.

Há duas situações a analisar neste trecho de código, a primeira observação a ser feita é no primeiro scanf, pois o argumento que lhe é passado é a string1, não &string1. A função scanf requer que os seus argumentos sejam ponteiros, associados a suas conversões, mas a string1 já corresponde a um apontador, pois é do tipo char*, logo não é necessário indicar o & da variável. (Damas, 1999)

Contudo é necessário colocar o & (endereço) no segundo scanf, pois estamos a utilizar a flag %a que aloca uma variável grande o suficiente para receber os caracteres e no fim devolve o apontador. As flags utilizadas neste pedaço de código, têm algumas particularidades especiais que devem ser tidas em conta. A flag %20 que é aplicada no primeiro scanf, restringe o buffer apenas a 20 caracteres, por outro lado a flag %a, realiza uma alocação dinâmica do input no momento em que este está a ser inseridos no buffer.

A segunda observação, consiste no resultado obtido quando se insere mais de 20 caracteres na primeira fase do programa. O primeiro scanf apenas lê os primeiros 20 caracteres, já o segundo recebe todos os caracteres que vierem depois dos 20 iniciais, sem esperar pela segunda parte do programa. Este problema deve-se ao scanf não ler linha a linha, mas sim ir recebendo os caracteres do stdin. Portanto o segundo scanf irá corresponder aos caracteres que excedem os 20 primeiros.

Algoritmo 1.1 : Código com proteção de Buffer-Overflow

```
#include <stdio.h>

int main()
{

  char *string1, *string2;

  string1 = (char *) malloc (25);

  puts ("Introduzir uma string com 20 ou menos caracteres.");
  scanf ("%20s", string1);
  printf ("\nString Introduzida:\n%s\n\n", string1);

  puts ("Introduzir string de qualquer tamanho.");
  scanf ("%as", &string2);
  printf ("\nString Introduzida:\n%s\n", string2);

  return 0;
}
```

No pedaço de código seguinte é possível observar o mesmo programa, mas sem as pequenas proteções nos scanfs, tornando assim o programa vulnerável a buffer-overflows.

Algoritmo 1.2 : Código sem proteção de Buffer-Overflow

```
#include <stdio.h>

int main()
{

  char *string1, *string2;

  string1 = (char *) malloc (25);

  puts ("Introduzir uma string com 20 ou menos caracteres.");
  scanf ("%s", string1);
  printf ("\nString Introduzida:\n%s\n\n", string1);

  puts ("Introduzir string de qualquer tamanho.");
  scanf ("%a", &string2);
  printf ("\nString Introduzida:\n%s\n", string2);

  return 0;
}
```

1.2.7.3. Buffer-OverFlow na Heap

De modo a compreender como funcionam os ataques de B.O na heap, é fundamental compreender como funciona a gestão da heap.
Em C existe uma biblioteca, a glibc que gere a heap como um conjunto de blocos. O tamanho dos blocos pode ser diferente, mas estão organizados sequencialmente, uma vez que a heap é uma zona circunjacente de memória. (Correia, 2010)

Os blocos disponíveis por uma questão de eficácia, fazem parte de uma lista duplamente ligada. Quando é necessário realizar uma reserva de um ou mais blocos, estes são removidos da lista de blocos livres e marcados como ocupados.
Todos os blocos são compostos por um cabeçalho com a informação de controlo, que inclui o estado do bloco (disponível/ocupado), o tamanho do bloco anterior, assim como o seu próprio tamanho. Um B.O é possível na heap devido ao facto de os dados estarem armazenados juntamente com informações de controlo, elevando assim os danos causados.

O funcionamento da glibc na libertação de espaço depende do estado dos blocos adjacentes ao que vai ser libertado. Caso o bloco a ser libertado esteja entre dois blocos ocupados, o libertado é inserido na lista dos blocos livres, através da atualização dos dois apontadores. Caso algum dos blocos adjacentes esteja livre, o bloco libertado funde-se com o que estivere livre. A sequência de passos para este processo consiste em três fases:

1ª fase - remoção dos blocos livres da lista duplamente ligada;
2ª fase – fusão dos blocos, atualização das informações do novo bloco; e
3ª fase – inserção do novo bloco na lista duplamente ligada.

Uma vulnerabilidade de BO na heap, quando examinada, permite ao atacante escrever nos blocos da heap, localizados em endereços acima do buffer vulnerável.

Na figura 1.12 é possível verificar um bloco (A) que contém uma vulnerabilidade, o atacante apenas tem que alterar os apontadores do outro bloco (B), que está seguido ao bloco (A) marcando (B) como livre, mesmo que este esteja ocupado. Quando o Bloco (A) for libertado, segundo o algoritmo da biblioteca glibc, este irá fundir o Bloco (A) com o Bloco (B), pois (B) é adjacente a (A). Mas antes da fusão o bloco (B) será removido da lista duplamente ligada. A partir deste momento o atacante pode escrever numa zona arbitrária da memória, pois este tem o controlo dos dois apontadores do Bloco B.

A partir deste momento o atacante pode provocar alterações no fluxo do programa, escrevendo o endereço de funções de biblioteca, ou inclusive correr código que está previamente carregado na heap.

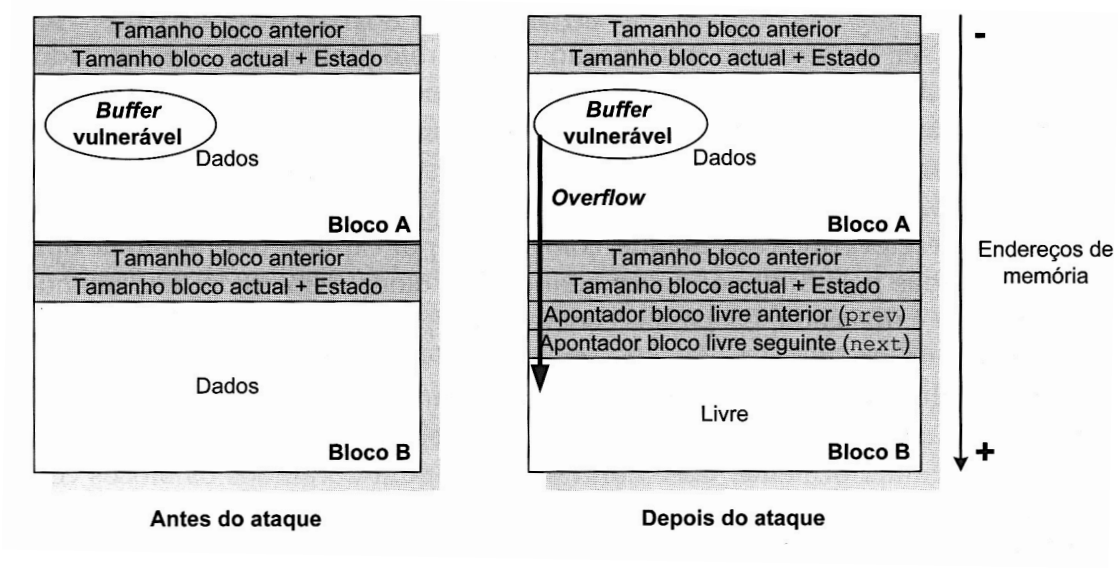


Fig. 1.12 - Ataque de Buffer-Overflow na Heap

1.2.8. Race Condition

Nos Sistemas Operativos, os processos que correm juntos partilham um espaço de memória, em que cada um pode ler ou escrever . (Tanenbaum, 2001)

Uma corrida ocorre quando numa aplicação é possível infringir um pressuposto de atomicidade, desta forma violando a ordem de um procedimento previamente programado. Denomina-se por janela de vulnerabilidade ao intervalo de tempo que permite fazer essa infração. (Correia, 2010)

Um exemplo de uma corrida pode ocorrer no print spooler. Quando um processo pretende imprimir um ficheiro, este coloca o nome do ficheiro num diretório especial, o diretório spooler. Outro processo, o printer daemon, procura regularmente se existe algum ficheiro para ser impresso, em caso positivo o processo imprime o ficheiro e remove-o do diretório de spooler.

Presume-se neste texto que o diretório de spooler possui um numero elevado de slots capazes de armazenar o nome do ficheiro, e admite-se também que existem duas variáveis partilhadas (**out** que aponta para o próximo ficheiro a ser impresso, **in** que aponta para a próxima slot vazia).

Em certo instante as slots de 0 a 3 estão disponíveis e as slots 4 a 6 estão ocupadas. No mesmo instante, o processos A e B decidem colocar em espera um ficheiro, então o processo A irá ler na variável in e guardar na slot 7, numa variável local denominada de next_free_slot. Nesse preciso instante ocorre uma falha no clock do sistema e o CPU decide que o processo A está a demorar muito tempo, então troca para o processo B.

O processo B irá ler a variável in que levará à slot 7, então o processo B irá atualizar a next_free_slot e colocar o nome do ficheiro na posição 7.

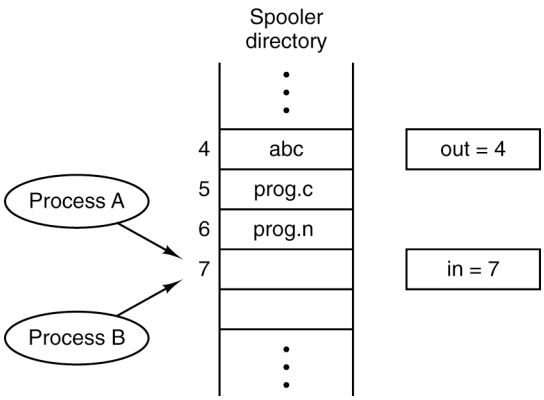


Fig. 1.13 - Dois processos a tentarem aceder à mesma memória partilhada

Mais tarde quando o processo A voltar a correr, irá ver que a posição livre é na slot 7, então vai escrever por cima do ficheiro do processo B, e vai incrementar uma posição na next_free_slot,. A consistência do sistema está mantida, mas o processo B nunca terá o seu ficheiro impresso.

A chave para prevenir problemas na partilha da memória, partilha de ficheiros, é proibir o acesso de vários processos à memória partilhada, no mesmo instante. Deve ser aplicada uma exclusão mútua, ou seja quando um processo está a aceder uma variável partilhada ou ficheiro, os outros processos estão excluídos de fazer o mesmo.

Na maioria do tempo, um processo está ocupado fazendo computação interna que não leva a corridas, mas quando o processo precisa de aceder a memória partilhada, este entra na região critica. De modo a reduzir o risco de corridas devem ser adoptadas algumas regras:

- Dois processos não podem estar simultaneamente na sua região critica
- Não se devem fazer suposições sobre a velocidade ou número de CPU's
- Nenhum processo fora da sua região crítica pode bloquear outro
- Nenhum processo deve esperar eternamente para entrar na sua zona crítica

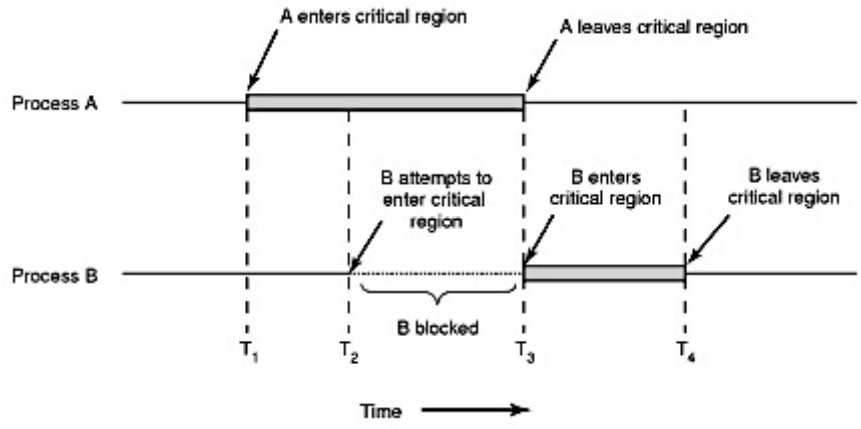


Fig. 1.14 - Exclusão mútua utilizando regiões críticas

1.2.9. DLL Injection

Outra das vulnerabilidades existentes em alguns Sistemas Operativos é a DLL Injection, ou em português Injeção de DLL. Esta é uma técnica que utiliza o espaço de endereçamento de um processo para executar código e esse código obriga o processo a carregar uma DLL.

Para melhor compreender este tipo de ataque é necessário compreender o que é uma DLL. Uma DLL é uma biblioteca que inclui código e dados que podem ser utilizados por mais do que uma aplicação ao mesmo tempo. (Support Microsoft, 2007)

Por exemplo a DLL de Comdlg32, que se encontra nos Sistemas Operativos Windows, é responsável pelas funções das caixas de diálogo. Devido a esta DLL, qualquer aplicação pode utilizar as suas funções para implementar uma caixa de diálogo, o que representa e promove a reutilização do código e a utilização eficaz da memória.

Um atacante, ao tentar injetar uma DLL, procura influenciar o comportamento de uma aplicação de maneira diferente à pretendida pelos seus programadores.

Um dos ataques mais comuns em que se utiliza as DLL é a intercepção de chamadas a funções.

As duas técnicas mais utilizadas para provocar um carregamento anormal de uma dll num programa é:

1. Procura-se o pid do processo alvo utilizando uma procura pelo task manager, sabendo o nome do processo pretendido;

2. Aloca-se alguma memória no processo alvo, e nesse local escreve-se o nome da DLL que vai ser injetada;

3. Uma nova thread é adicionada no processo alvo com o endereço de LoadLibrary e como argumento o endereço ajustado para o endereço da memória alocada previamente; e

4. O sistema agora irá carregar automaticamente a nova DLL.

Algumas funcionalidades dos Sistemas Operativos que são geralmente usadas para facilitar um ataque por DLL injection são:

Hooks

Um Hook é uma função que se pode criar como parte de uma dll ou num aplicativo para monitorizar os "acontecimentos dentro do sistema operativo do Windows. A ideia é escrever uma função que é chamada cada vez que um determinado evento ocorre no Windows - por exemplo quando um utilizador pressiona uma tecla no teclado ou move o rato.

Os Hooks foram fornecidos pela Microsoft principalmente para ajudar os programadores a realizarem Debug às suas aplicações, mas pode ser colocada em uso em muitas maneiras diferentes.

Existem dois tipos de Hooks o global ou local:

- Um Hook local é aquele que monitoriza os eventos apenas por um programa específico (ou thread); e

- Um Hook global monitoriza todo o sistema.

Ambos os tipos de Hooks são configurados da mesma maneira, a diferença principal é que para um hook local, a função a ser chamada pode estar dentro do programa que está a monitorizar, mas com um hook global a função deve ser armazenada e carregada a partir de uma dll separada.

Existem alguns métodos conhecidos que permitem a injeção de uma DLL num processo, pois são utilizadas no hacking de jogos, no keygenning, etc.

Dois dos métodos mais utilizados são:

- CreateRemoteThread; e
- SetWindowsHookEx.

CreateRemoteThread

Esta metodologia é bastante simples e até elegante de realizar, a API do Windows fornece-nos uma função chamada CreateRemoteThread (). Esta função permite iniciar uma thread em outro processo, se resultar, a função irá retornar o handle para a nova thread. A principal desvantagem deste método é que irá funcionar apenas no Windows NT e superiores.

A definição da função é:

```
HANDLE WINAPI CreateRemoteThread(
__in   HANDLE hProcess,
__in   LPSECURITY_ATTRIBUTES lpThreadAttributes,
__in   SIZE_T dwStackSize,
__in   LPTHREAD_START_ROUTINE lpStartAddress,
__in   LPVOID lpParameter,
__in   DWORD dwCreationFlags,
__out  LPDWORD lpThreadId
);
```

SetWindowsHookEx

Este método é um pouco mais intrusivo do que o primeiro, e cria uma perturbação no processo onde foi aplicada a injeção, o que não é desejo do atacante. No entanto, é um pouco mais fácil de utilizar que o primeiro e tem mais vantagens face ao primeiro método. A função SetWindowsHookEx () esta preparada para permitir que o utilizador faça "hook" de mensagens para determinada thread. Para aproveitar esta vantagem é necessário que o atacante consiga injectar a dll no espaço de endereço do processo. A dll deve ter uma função para o "hook" criado, caso contrário, será crashar o processo.

```
HHOOK WINAPI SetWindowsHookEx(
  __in  int idHook,
  __in  HOOKPROC lpfn,
  __in  HINSTANCE hMod,
  __in  DWORD dwThreadId
);
```

1.2.10. Classificação de software malicioso

Existe uma gama de possibilidades de se vulnerabilizar um computador, as técnicas mais usadas são através de e-mails, arquivos partilhados e páginas da web infectadas. As consequências também são bastante variadas, algumas têm como objetivo infectar computadores alheios para, em seguida, danificar os seus elementos, seja excluindo arquivos, seja alterando o funcionamento da máquina ou até mesmo deixando o computador vulnerável a outros tipos de ataques. Porém existem aqueles que não visam prejudicar a máquina, mas sim, o seu utilizador, como os softwares que têm como objetivo capturar informações sigilosas, como passwords e números de cartões de crédito para repassá-las para terceiros, causando graves transtornos às vítimas.

Os tipos de software malicioso são usualmente classificados da seguinte forma:

Malware

Termo geralmente aplicado a qualquer software desenvolvido para causar danos em computadores. Estão incluídos vírus, cavalos-de-tróia e worms.

Vírus

Pequenos programas criados para causar danos na máquina infectada, apagando dados, capturando informações ou alterando o funcionamento do computador. O nome vem da grande semelhança que estes programas têm com os vírus biológicos, pois, depois de infectar um computador, este instala-se num um programa e usa-o como base para se multiplicar e se disseminar para outros computadores.

Podem incorporar-se a quase todo o tipo de arquivos. Podem mostrar apenas mensagens ou imagens, sem danificar arquivos da máquina infectada, mas podem ir consumindo a capacidade de armazenamento e de memória ou diminuindo o desempenho do computador infectado.

Têm a capacidade de destruir ficheiros, formatar o disco rígido, ou até a destruição total do sistema operativo.

Os utilizadores dos sistemas operativos da Microsoft são as principais vítimas dos vírus, já que estes sistemas são os mais utilizados no mundo, existindo para este sistema os mais variados tipos diferentes de vírus. Existem também vírus para os sistemas operativos Mac OS, mas estes são extremamente raros e bastante limitados. Até algum tempo atrás, a contaminação era feita através da partilha de arquivos em disquete, mas agora a internet é o seu principal meio de propagação, podendo contaminar milhares de computadores em poucos minutos. Os métodos mais comum são através de e-mails, chats e páginas html infectadas.

Worms

Programa que se auto-replica, assim como os vírus, porém a principal diferença entre eles é a forma de propagação, os worms podem se propagar rapidamente para outros computadores, pela internet ou por redes locais, fazendo cópias de si mesmo em cada computador. É um programa completo, não precisando de uma base para entrar em ação, como o vírus. Primeiro o worm, controla os recursos que permitem o transporte de arquivos e informações, depois o este desloca-se sozinho para outros computadores. O grande perigo é sua enorme capacidade de se replicar. Pode ser projetado para fazer muitas coisas, como, por exemplo, excluir arquivos do sistema, enviar documentos por e-mail ou podem provocar danos apenas com o tráfego de rede gerado pela sua reprodução em massa.

Cavalo-de-Tróia

Também chamado de Trojan Horse, ou apenas Trojan, este programa, diferente dos vírus e dos worms, não se duplica, alguns são programados para se autodestruírem após algum tempo ou com algum comando do cliente. A infecção ocorre através de arquivos anexos a e-mails, mensagens instantâneas ou downloads. O programa é quase sempre uma animação ou um conjunto de imagens pornográficas, mas é durante a exibição dessas imagens que o computador é infectado.

Os cavalos-de-Tróia são divididos em duas partes: o servidor e o cliente. O servidor geralmente fica oculto em algum arquivo, que, quando executado, permite que o servidor seja instalado no computador da vítima, sem que esta saiba. Daí por diante o atacante passa a ter controle do computador infectado.

Spyware

Este é um programa automático de computador que recolhe informações sobre o utilizador e repassa essa informação para uma entidade externa na internet que não tem como objetivo, a manipulação do sistema do utilizador. Ao invés, permanece despercebido no sistema. Este pode ser obtido por download de um site, mensagens de e-mail, mensagens instantâneas e conexões diretas para o partilha de ficheiros, podendo também conter vírus.

1.2.11 Métodos de propagação

A Microsoft, de modo a entender melhor a propagação do malware, realizou uma análise de modo a verificar os riscos que os clientes correm a partir do momento em que um malware é lançado. Quando um novo malware é divulgado, os antivírus não possuem ainda uma assinatura que permita a detecção e remoção deste, assim, como o sistema operativo ainda não tem a atualização que permite eliminar a vulnerabilidade que permitiu o ataque. Muitos dos profissionais da segurança devem ter em conta o método e a forma como o malware se propaga, pois podem assim gerir os riscos. Os departamentos de TI, enfrentam restrições de orçamento, tempo, pessoal e de recursos, tendo conhecimento sobre esta problemática, podem estabelecer prioridades nas defesas e ajudar a manter as redes, softwares e pessoas seguras.

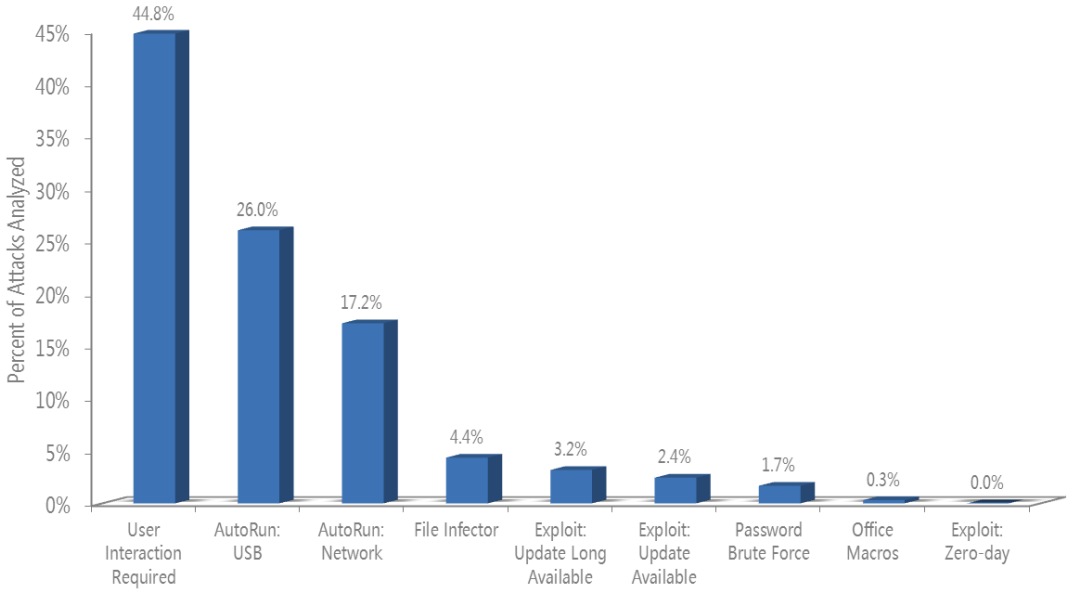


Fig. 1.15 - Métodos de propagação de malware

Os diferentes métodos de propagação de malware referenciados na figura acima são:

- **User Interaction Required**. Quando um utilizador tem de realizar uma ação para que o computador fique comprometido;

- **AutoRun: USB**. Esta ameaça tem como vantagem o sistema de arranque do Windows para infectar drives removíveis;

- **AutoRun: Network**. Esta ameaça tem como vantagem o arranque automático de um computador numa rede, permitindo infectar os discos do computador;

- **File Infector**. A ameaça espalha-se pela modificação de ficheiros .exe reescrevendo ou juntando código perigoso;

- **Exploit: Update Long Available**. O representante lança uma atualização de segurança para eliminar a vulnerabilidade, um ano ou mais antes do ataque;

- **Exploit: Update Available**. O representante lança uma atualização de segurança para eliminar a vulnerabilidade, um ano ou menos antes do ataque;

- **Exploit: Zero-day**. O representante não lançou uma atualização de segurança para eliminar a vulnerabilidade;

- **Password Brute Force**. A ameaça espalha-se através da realização de força bruta nas passwords dos volumes das máquinas com recurso à rede;

- **Office Macros**. A ameaça espalha-se através da infeção de documentos do Office com código malicioso;

Entre a variedade de mecanismos técnicos e não técnicos que as pessoas mal-intencionadas têm à sua disposição para atacar computadores e roubar informação, a exploração de uma vulnerabilidade Zero-day representa a exploração da vulnerabilidade antes do fornecedor lançar uma atualização de segurança. Uma vulnerabilidade Zero-day pode surgir a qualquer momento, deixando, independentemente da segurança, o sistema à mercê da vulnerabilidade. Algumas tecnologias, como a DEP e a ASLR, foram introduzidas para tornar mais complicada a exploração de software, mas as vulnerabilidades Zero-day continuam a ser capazes de criar problemas.

A vulnerabilidade Zero-day é especialmente alarmante para os consumidores e profissionais de TI. Além de perigosa, combina o medo do desconhecido e a incapacidade para corrigir o problema, o que deixa os utilizadores e administradores indefesos perante o ataque. É assim

natural que uma vulnerabilidade deste tipo receba uma cobertura considerável pela comunicação social quando surge e é muitas vezes tratada com nível máximo de urgência por parte do fornecedor afectado.

1.3. Âmbito e Objectivos da Dissertação

Atualmente os sistemas distribuídos são utilizados de forma generalizada, o que implica uma constante comunicação entre diferentes computadores e entre diferentes sistemas operativos, perante este cenário é possível deduzir que existe muita informação a circular nas redes de computadores.
Muitas máquinas a comunicar entre si implicam mais risco e vulnerabilidades na proteção da informação, desta forma, um dos motivos que levou à elaboração desta dissertação foi conhecer as principais vulnerabilidades existentes na memória, os mecanismos de proteção em diferentes sistemas operativos, tipos de ataques existentes e testar as proteções existentes. Pretende-se também demonstrar formas de ataque possíveis, com especial ênfase para os ataques combinados de engenharia social com exploração de vulnerabilidades técnicas, e efetuar recomendações no sentido de melhorar os sistemas de segurança organizacionais.

De modo a demonstrar que os atuais sistemas operativos, não são 100 % eficazes, pois estão sujeitos a vulnerabilidades, pretende-se:

- Demonstrar que é possível realizar um ataque híbrido (engenharia social e técnico) capaz de produzir resultados que contornam algumas das proteções que geralmente debelam ataques quer de engenharia social quer técnicos; e
- Apontar vias para melhorar a resistência a este tipo de ataques híbridos.

2. Estado da Arte

Devido à importância dos problemas acima apresentados, têm sido pesquisadas novas metodologias de defesa para as vulnerabilidades detectadas nos sistemas operativos.

2.1. Formas de proteção

Escrita de código seguro:

Não é fácil escrever código seguro, é uma conduta que poucos programadores conseguem seguir tendo em conta que o fazem segundo a Linguagem C, que promove a performance em vez da segurança.
Um dos processos mais acessíveis produzir códigos mais seguros é substituir algumas funções por outras equivalentes que são mais seguras.
Independentemente dos esforços realizados, as falhas de buffer-overflow podem ser complicadas de eliminar.

Função	Risco	Solução
gets()	Extremo	Usar fgets(buffer,tamanho,stdin)
strcpy()	Alto	Usar strncpy() ou stlcpy()
strcat()	Alto	Usar strncat() ou strlcat()
sprintf()	Alto	Usar snprintf()
scanf()	Alto	Utilizar especificadores para limitar o tamanho ou analisar a entrada de dados
getc()	Moderado	Ao utilizar esta função loop, verificar o buffer destino para que este não rebente
fgets()	Baixo	Verificar se o tamanho do destino suporta o argumento da função
snprintf()	Baixo	Verificar se o tamanho do destino suporta o argumento da função

Tabela 2.1 - Algumas funções vulneráveis a ataques de buffer-overflow

Proteção do Sistema Operativo:

Existem modificações feitas no Kernel de um SO que apontam para melhorias na segurança do sistema. O objectivo é transformar o segmento de dados e a pilha do espaço de endereçamento de um programa vitima em espaço não-executável, para não permitir que os atacantes consigam correr o código que foi injetado no buffer.

Proteção em tempo de Compilação:

Uma proteção em tempo de compilação é feita através da comprovação dos limites dos vectores. Esta verificação elimina totalmente os ataques e vulnerabilidades. Essa previne ultrapassar o tamanho dos buffers.

2.1.1. Modelos de proteção do Windows

Controlos de Acesso

Os sistemas operativos da Microsoft oferecem bastantes meios que permitem controlar o acesso ao sistema. O mecanismo de controlo de acessos da Microsoft mais conhecido por Access Control List, ou ACL é uma parte fundamental dos Sistemas Operativos Windows.

As ACL´s são listas de tuplos com a identidade de um utilizador e as suas permissões sobre o recurso que lidera a lista. Reciprocamente as capacidades são listas de tuplos com o nome do recurso e as permissões do utilizador de encabeça a lista, ou seja uma ACL é um mecanismo de controlo de acesso que determina o nível de acesso que uma conta a um determinado recurso.

- **Discretionary access control lists (DACLs)**: As DACLs identificam os utilizadores aos quais está atribuído ou negado o acesso a determinado recurso. Se a DACL não identificar explicitamente um utilizador ou grupo onde o utilizador esteja incluído, o acesso ao recurso é negado. Por omissão a DACL é controlada pelo utilizador que criou o recurso.

- **System access control lists (SACLs)**: As SACLs identificam os utilizadores ou grupos que devem ser auditados quando acedem com ou sem sucesso ao recurso. A auditoria é utilizada para monitorizar os eventos relacionados com os sistema ou com a segurança da rede, de modo a encontrar falhas de segurança e determinar a extensão e localização do problema. Por omissão as SACL é controlada pelo utilizador que criou recurso

  .

Mecanismos de proteção da Microsoft

Com o passar dos anos a Microsoft foi aplicando proteções nos seus sistemas operativos com o objectivo de diminuir as vulnerabilidades encontradas em algumas aplicações (Sotirov, 2008).

	XP SP2/SP3	Vista SP0	Vista SP1	7
/GS				
stack cookies	Yes	Yes	Yes	Yes
variable reordering	Yes	Yes	Yes	Yes
Heap protection				
safe unlinking	Yes	Yes	Yes	Yes
safe lookaside lists	No	Yes	Yes	Yes
heap metadata cookies	Yes	Yes	Yes	Yes
heap metadata encryption	No	Yes	Yes	Yes
DEP				
permanent DEP	No	No	Yes	Yes
OptOut mode by default	No	No	No	Yes
ASLR				
PEB/TEB	Yes	Yes	Yes	Yes
heap	No	Yes	Yes	Yes
stack	Yes	Yes	Yes	Yes
images	No	Yes	Yes	Yes

Tabela 2.2 – Algumas mecanismos de proteção da Microsoft

2.1.1.1. /GS

Stack Cookies:

Uma das opções do compilador do Visual C++ é a possibilidade de detectar buffers overflows na pilha. Se essa opção estiver ativa permite que o compilador guarde na pilha um valor aleatório entre as variáveis globais e o endereço de retorno da função.

Este valor é denominado por Stack Cookie, caso um atacante tente explorar um ataque de buffer overflow e tentar reescrever sobre o endereço de retorno da função irá reescrever também sobre o Stack Cookie.
Essa mudança será detectada pelo fim da função, fazendo com que o programa aborte antes de chegar ao endereço de retorno.

Variable Reordering:

Uma das limitações da proteção de /GS deve-se ao facto de apenas ser possível detectar falhas de buffer oveflow no momento em que a função retorna o Stack Cookie reescrito. Para prevenir o atacante de reescrever variáveis locais utilizadas na função, o compilador modifica a disposição da pilha, reorganiza as variáveis, aplica buffers de strings em endereços mais altos que os das variáveis locais. Isto garante que um buffer overflow não consiga reescrever sobre as variáveis locais. Os argumentos de funções que possam conter apontadores, estão protegidos por serem alocados num espaço extra na pilha e copiados depois do valor das variáveis locais.
Os valores originais encontram-se alocados depois do endereço de retorno da função e não são utilizados no código restante.

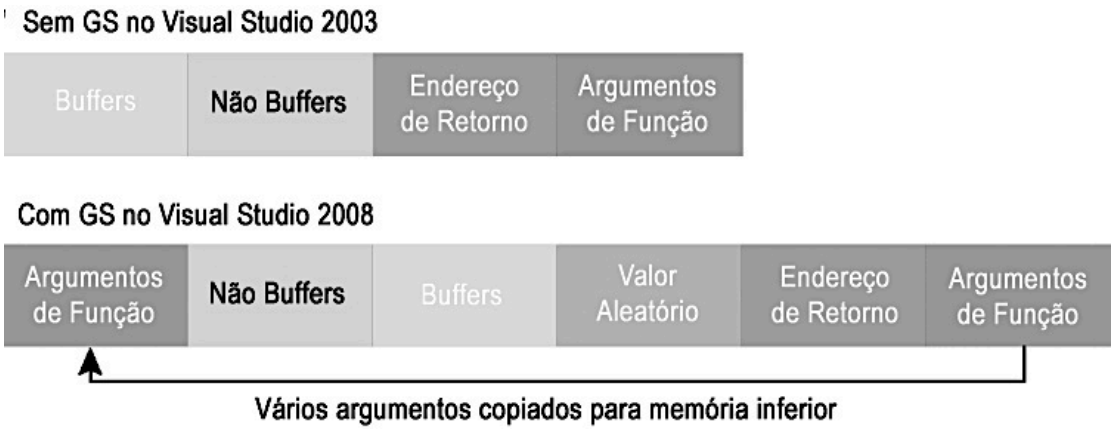


Fig. 2.1 – Implementação de GS no Visual Studio 2003

2.1.1.2. Heap Protection

Um meio de exploração comum para heap overflows em versões antigas do Windows era reescrever o cabeçalho de uma fração da heap e criar um bloco livre falso com apontadores

flink e blink controlados pelo atacante. Quando o bloco livre é alocado ou une-se a outros blocos o alocador irá escrever o ponteiro flink no endereço apontado pelo blink. Isto permite que o atacante consiga uma escrita arbitrária de 4 bytes num local qualquer da memória o que permite correr código Shell.
Os mecanismos de proteção heap no Windows XP SP2 e Windows Vista são projetados para parar esta técnica exploração.

Safe unlinking:

A partir do Windows XP SP2, o alocador da heap realiza um safe unlinking ao remover pedaços da lista livre. Antes de utilizar os ponteiros flink e blink, este verifica se tanto o flink->blink e o blink->flink apontam para o mesmo bloco na heap. Isto previne que o atacante aponte o flink ou o blink para um local arbitrário na memória.

Heap metadata cookies and encryption:

Como adição ao safe unlinking o alocador do XP SP2 guarda um cookie com um byte na heap de cada fração da pilha. Se o cabeçalho do bloco da pilha for reescrito então o cookie não irá corresponder portanto o alocador irá verificar que a pilha esta corrompida.
No Windows Vista o cookie é complementado por uma encriptação dos metadados da pilha. Todos os campos da pilha são encriptados segundo um valor aleatório de 32 bit e são desencriptados antes de serem utilizados. Os cookies e a encriptação dos metadados são bastante eficazes em prevenir que o atacante reescreva no cabeçalho de um bloco ou crie falsos blocos na heap e na pilha.

2.1.1.3.DEP

Data Execution Prevention, é um meio de proteção que proíbe a execução de código em páginas de memória marcadas como não executáveis. Ativando o DEP previne-se que o atacante consiga executar código na pilha. Caso este meio esteja ativado e um programa tente executar código de páginas não executáveis é levantada uma violation exception e o programa irá abortar.

Permanent DEP:

Devido a problemas de compatibilidade entre programas e a DEP, esta não está ativada por omissão em todos os processos do sistema. O administrador do sistema pode optar por quatro configurações para a DEP, configurações que se encontram no ficheiro boot.ini no Windows XP ou na configuração Boot no caso do Windows Vista.

OptIn

Esta é a configuração por omissão presente no XP, Vista e no Seven. Nesta situação a proteção é aplicada apenas para processos do sistema. Todos os outros processos estão fora da proteção DEP.

OptOut

Todos os processos estão protegidos pela DEP, excluindo apenas os que o administrador coloca numa lista de exceção ou os que não são compatíveis com esta proteção

AlwaysOn

Todos os processos estão sob a proteção DEP

AlwaysOff

Nenhum processo está sob a proteção DEP

2.1.1.4. ASLR

Um recurso que realiza a aleatoriedade dos endereços dos objetos mapeados na memória virtual de um determinado processo é o ASLR que representa Address Space Layout Randomization. Quando é implementado corretamente revela ser uma dificuldade para o atacante pois este não sabe a localização exata do endereço que quer reescrever. A proteção por ASLR só está presente nativamente no Windows Vista e no Seven.

Image Randomization

O posicionamento aleatório de imagens tem como objectivo colocar as imagens num local aleatório do espaço de endereçamento da memória virtual de cada processo. No caso do

ASLR do Vista, tem a capacidade de colocar aleatoriamente a posição das dll´s como os executáveis.

2.1.1.5. DLL Randomization

Uma dll é carregada no mesmo endereço que o processo que a utiliza, para permitir que a memória utilizada pela dll seja partilhada. Para facilitar este procedimento, um bitmap global chamado MilmageBitMap é usado para representar o espaço de endereços de 0x50000000 a 0x78000000. O bitmap tem 0x2800 bits de comprimento em que cada bit representa 64KB de memória. À medida que cada DLL é carregada, a sua posição é gravada por definição com os bits apropriados no bitmap para marcar a memória onde a DLL é mapeada. Quando a mesma DLL é carregada em outro processo, o seu objeto de seção é reutilizado e é mapeado nos mesmos endereços virtuais.

Heap randomization

As aplicações do Windows muitas vezes utilizavam várias heaps, cada uma criada pela função RtlCreateHeap. No passado na criação de uma heap era feita uma procura de espaço de endereços linear, a partir de um ponto escolhido pelo chamador. Alocar uma heap com este método não garantia segurança pois permitia uma previsão da sua localização.
No Vista e no Seven foi adicionada alguma aleatoriedade ao processo de alocação da heap, com o objectivo de dificultar o sucesso de um possível atacante. Esta aleatoriedade ocorre durante o inicio da RtlCreateHeap um valor de 5 bits aleatórios é gerado e multiplicado por 64k, este valor é então utilizado como deslocamento do endereço base, onde a estrutura de dados irá começar. O bloco que se encontra antes deste será libertado posteriormente.

2.1.2. Modelos de segurança Mac OS X

Os serviços de segurança do Mac OS X são construídos sob dois standard open-source, começando pelo Berkeley Software Distribution (BSD), uma forma de UNIX que fornece serviços fundamentais, incluindo o sistema de ficheiros e permissões de acesso aos mesmos. Outro standard open-source é a CDSA, que representa Common Data Security Architecture este open-source fornece um conjunto de serviços de segurança, incluindo permissões de

acesso mais específico, autenticação da identidade do utilizador, criptografia e armazenamento seguro de dados (Mac OS X Security Configuration, 2010)..

O Kernel do Mac OS X é construído com base em BSD, que fornece o sistema básico de ficheiros e rede, esquema de identificação de utilizadores. Este sistema impõe restrições de acesso a arquivos e recursos do sistema, baseados em ID's de utilizador.

O Mach também faz parte do Kernel do Mac OS X, fornece o gestor de memória, controlo de threads, abstração de hardware e a comunicação entre os processos.

2.1.2.1. Permissões de Acesso

Um dos aspectos mais importantes da segurança de computadores é a concessão ou negação de acesso, muitas das vezes designado por direitos de acesso. A permissão é a capacidade de executar uma operação específica, como o acesso a dados ou a execução de código. As permissões são concedidas ao nível das pastas, subpastas, ficheiros, aplicações, dados específicos ou funções das aplicações.

No Mac OS X as permissões são controladas segundo vários níveis, desde os componentes Mach e BSD do Kernel, aos níveis mais elevados do sistema operativo.

2.1.2.2 Controlo de acesso Obrigatório

O Mac OS X utiliza um mecanismo de controlo no acesso, conhecido por controlo de acesso Obrigatório (Mandatory Access Control – MAC). Este mecanismo não é visível para o utilizador, mas está incluído no sistema para proteger o computador. O controlo de acesso obrigatório é uma política que não pode ser substituída, pois define as restrições ao software que são criadas pelo desenvolvedor do software. Esta tecnologia ajuda a permitir vários recursos importantes, incluindo sandboxing, controlos parentais e preferências de gestão.

O sandboxing é um sistema presente no Mac OS que ajuda a garantir que o sistema faz apenas o que está destinado a fazer isto é garantido aplicando restrições no acesso das aplicações. As restrições são ao nível do acesso a arquivos, à rede e ao lançamento de outras aplicações. No Mac OS X, muitas aplicações do sistema de ajuda que comunicam com a rede são protegidas pelo sandboxing para impedir o acesso por parte de um atacante.

O sandboxing baseia-se no sistema de controlo de acessos, que é implementado ao nível do Kernel cada aplicação tem um perfil de sandboxing e cada perfil é composto pelos recursos que são acessíveis.

2.1.2.3. Proteção de memória em Runtime

O Mac OS X, a correr num sistema de 64 bits, suporta memória e proteção executável. Esta proteção evita alocações de memória ou execuções de código arbitrário forçadas no processador. O Mac OS X tem as seguintes características num sistema de 64 bits: não-execução de pilha, não-execução de dados e não-execução da heap. A proteção de não-execução de pilha também se encontra disponível para o sistema de 32 bits, mas num sistema de 64 bits a proteção é completa, pois a proteção de execução de código é feita tanto na heap e na pilha.

O sistema operativo inclui o sistema de aleatoriedade de posições de memória, sempre que o sistema inicia as posições de memória são diferentes. Isto não permite que o invasor saiba o endereço onde os processos do sistema se encontram.

2.2.2. Proteções contra ataques informáticos

Duas formas mais comuns de instalar malware no sistema operativo é através de agregação de software e engenharia social. Através de pacotes de software, o malware é compactado junto de utilitários de software. Muitas das vezes o utilizador não tem noção do perigo que corre ao instalar um programa desconhecido e com a ajuda da engenharia social o utilizador é incitado a instalar o software. Normalmente o utilizador recebe um e-mail ou um pop-up do seu browser contendo instruções para abrir um arquivo ou para visitar um site.

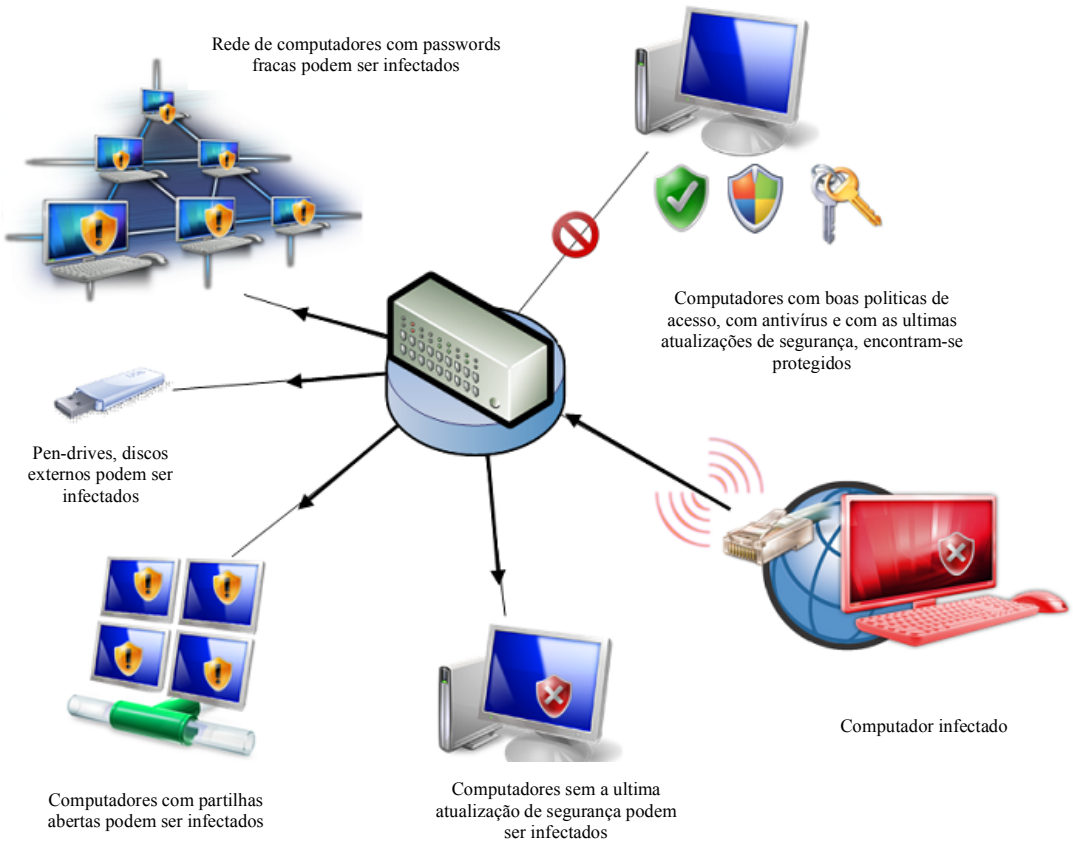


Fig. 2.2 – Meios de infeção por malware

O décimo relatório Security Intelligence da Microsoft, mostra um aumento do malware visando o Windows 7, que já se encontra instalado em cerca de um quarto de todos os computadores Windows. Embora tenha havido uma queda no malware visando o Windows XP, este continua a representar a maioria do malware existente para o Windows até hoje

Existem muitas razões para se gostar do Windows XP, mas a segurança não é um desses pontos, o sistema operativo ainda recebe algumas correções regulares e continuará a recebê-las até Abril de 2014, mas estas correções não representam o nível de segurança presente no Windows Vista. O que revela que o Windows XP é agora notavelmente menos seguro face ao momento do seu lançamento.

Com o aparecimento do Windows Vista e 7 surgiram novas proteções significativamente melhoradas para travar engenharia e agregação de software. Com as configurações padrão, o malware que tente instalar-se terá de contornar dois níveis de proteção: UAC e Windows Defender.

O contra-argumento é a sua simplicidade e a ausência do UAC (User Account Control) presente no Windows Vista e 7, que torna o uso deste dois últimos sistemas operativos um pouco incomodativos, pois são muito ávidos a avisar quando são realizadas alterações no sistema.
No entanto, o facto de limitar o que o malware pode realizar no computador, permite afirmar que o UAC torna o Windows Vista e 7 bastante seguros, pois o UAC funciona da seguinte forma:

- Se o utilizador é um utilizador limitado, todos os programas são executados como utilizador limitado. Se algum programa necessitar de gravar ficheiros em pastas globais do sistema ou configurações globais no registo, será necessário executá-lo como administrador. Isso pode ser feito clicando com o botão direito num programa e escolher “Executar como”.

- Se o utilizador é um administrador, até ao Windows Server 2003 todos os programas era classificados como administrador, tendo por isso acesso completo e sem restrições ao sistema. Agora no Windows Vista e 7, ocorre uma diferença, mesmo com conta de administrador, os programas executados pelos administradores são executados com privilégios limitados. Para executar um programa realmente com direitos de administrador, deve-se permitir e autorizar essa execução.

O UAC funciona como base para outros recursos úteis do SO, incluindo o modo protegido do IE, limitando rigorosamente e impedindo-o de escrever em certas zonas do disco rígido ou no registo do sistema sem a permissão do utilizador. Barras de ferramentas e outros add-ons herdam o mesmo nível de segurança, de modo que qualquer malware não seja capaz de se instalar, nem de instalar outros malwares.

O Windows 7 apresenta um UAC melhorado e com a possibilidade de ser personalizado, o que possibilita o utilizador de escolher em que circunstâncias pode ser alertado. Existem diferentes níveis de personalização o que permite desligar qualquer aviso feito pelo UAC, consentindo o trabalho do utilizador livre de distrações (Windows 7 vs. Windows Vista UAC, 2009).

Tasks Which Requires Administrator Privileges	Windows Vista UAC Prompt Trigger	Windows 7 UAC Prompt Trigger
Running an Application as an Administrator	Yes	Yes
Changes to files and folders in the Windows and Program Files folders	Yes	No
Installing applications	Yes	Yes
Uninstalling applications	Yes	No (in most cases)
Installing & uninstalling device drivers	Yes	Yes
Installing ActiveX controls	Yes	Yes
Changing settings for Windows Firewall	Yes	No
Changing UAC settings	Yes	Yes
Configuring Windows Update Settings	Yes	No
Adding or removing user accounts	Yes	No
Changing a user's account type	Yes	No
Configuring Parental Controls	Yes	No
Running Task Scheduler	Yes	No
Backup & Restore Files and Settings Using Backup & Restore or Windows Easy Transfer	Yes	No
Viewing or changing another user's folders and files	Yes	No
Running Disk Defragmenter	Yes	No
Changes to system-wide settings	Yes	No
Total	**17**	**5.5**

Fig. 2.3 – Windows 7 UAC vs. Windows Vista UAC

No entanto, o UAC ainda não é totalmente seguro, existem manuais documentados sobre o UAC envolvendo o Internet Explorer, escalamento de privilégios de segurança de tokens.

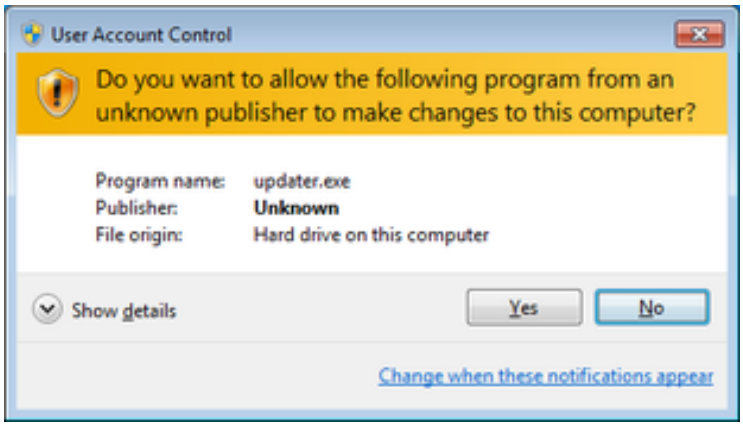


Fig. 2.4 – Janela de aviso UAC

O Windows Defender é uma proteção do Windows Vista e 7 que oferece proteção contra spyware e outros softwares potencialmente perigosos e indesejados. A assinatura do Windows Defender utiliza descrições que identificam spywares e outros softwares potencialmente indesejados para detectar e remover aplicações conhecidas. Regularmente o Windows Defender atualiza as suas assinaturas a partir da Microsoft para que este possa identificar e remover spyware recente e outros softwares.

Além disso, o Windows Defender inclui proteção em tempo real e monitoriza os locais críticos do sistema operativo, onde o malware costuma aplicar modificações. Esta proteção em tempo real rastreia a pasta de arranque Run, as chaves de Registo e outras áreas do sistema operativo. Se um programar tentar realizar uma alteração numa das áreas protegidas do sistema operativo, o Windows Defender solicita que o utilizador tome medidas adequadas. O Windows Defender também pode realizar uma análise sobre todo o sistema de modo a detectar e remover spyware conhecido.
Embora o Windows Defender realize proteção em tempo real para evitar a maioria das infeções, as atualizações permitem que as análises detectem e removam o malware recém-descoberto que poderia ter escapado às defesas da proteção em tempo real.

A comunidade SpyNet da Microsoft permite que o Windows Defender comunique descobertas sobre novas aplicações e receba as identificações feitas pelos utilizadores sobre possível malware. Dependendo de como o Windows Defender está configurado, pode fornecer um feedback para a comunidade SpyNet sobre novas aplicações e sobre as aplicações instaladas pelos utilizadores. O feedback da SpyNet ajuda a Microsoft e os utilizadores a distinguirem malware de software legitimo, permitindo que o Windows Defender melhore a sua precisão na identificação de malware e reduza o número de falsos alarmes. Em ambientes empresariais, os departamentos de TI geralmente lidam com a remoção do software. O Windows Defender também pode ser instalado no Windows XP com SP2.

O Internet Explorer visto ser o browser da Microsoft, já foi por diversas vezes citado como o mais vulnerável e pouco seguro e parte dessas ideias devem-se ao facto de ser o browser mais popular. Tendo isto em conta a Microsoft tem vindo a aplicar melhorias no seu browser (NSS Labs, 2010)..

Segundo o novo relatório pela NSS Labs, as duas últimas versões do IE são as que melhor protegem os utilizadores de malware derivado e engenharia social. Estes tipos de ameaças são particularmente difíceis de detectar, pois são bem implementadas e de modo a que o utilizador pense que o seu download está livre de qualquer falsificação maliciosa.

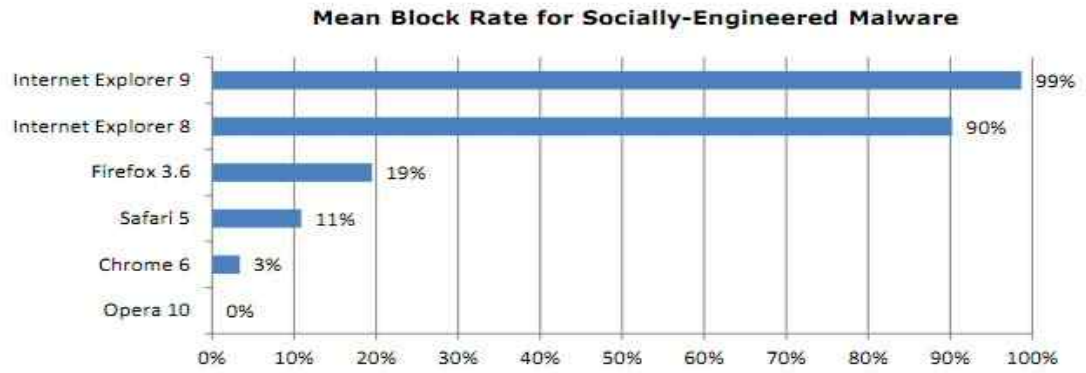


Fig. 2.5 - Detecção de malware por engenharia social

Os últimos testes da NSS Labs, demostraram que o Internet Explorer 8 e o IE 9, são os líderes absolutos na detecção de malware derivado de engenharia social. Enquanto o Internet Explorer 9 detecta e bloqueia 99%, o seu antecessor Internet Explorer 8, detecta e bloqueia 98% das ameaças. Comparativamente com o seu maior concorrente, verifica-se que o Firefox 3.6 detecta apenas cerca de 20%, combinando a percentagem do Firefox 3.6, Safari 5, Chrome8 e Opera 10 verifica-se que nem chegam a formar metade da proteção do IE 8.

Outro dos problemas atuais deve-se ao facto de muitos dos utilizadores não perceberem que o seu computador pode ficar infectado com apenas uma visita a um website aparentemente legitimo. No entanto mais de 80% dos URLs maliciosos são sites legítimos que foram adulterados por hackers. Isto é conseguido através da exploração de vulnerabilidades no software ou por roubar credenciais de acesso a máquinas infectadas por malware.

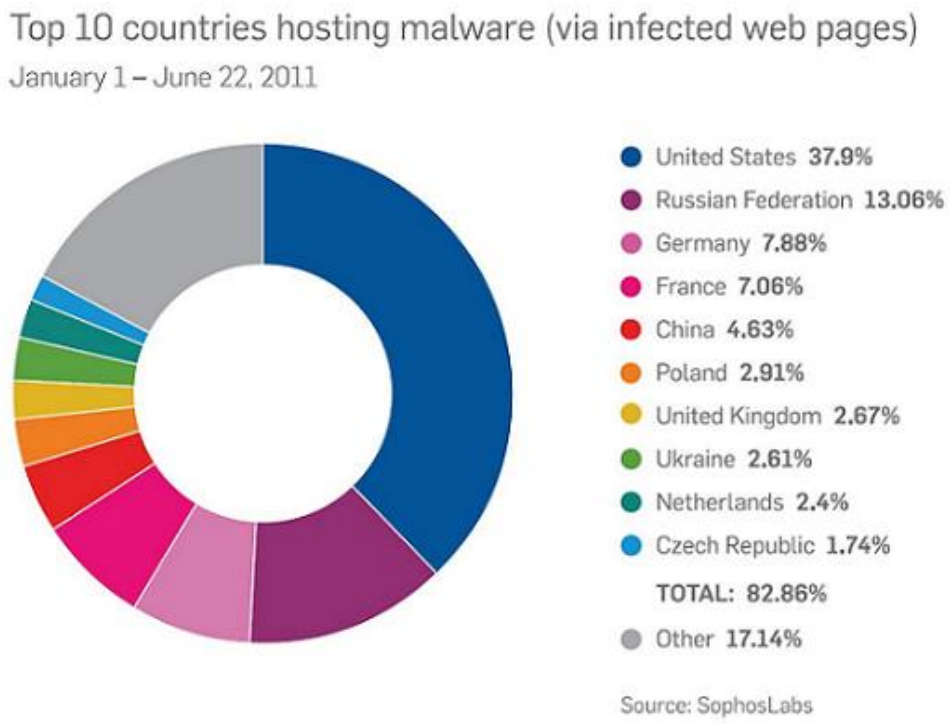


Fig. 2.6 – Top 10 dos países que alojam malware (via websites infectados)

Os Estados Unidos ainda detém o primeiro lugar na lista dos países que hospedam malware, embora a percentagem total de malware hospedado por eles tenha diminuído ligeiramente

durante o primeiro semestre de 2011, descendo 1,4 pontos de 39,39% em 2010. A Russia ocupava em 2011 o segundo lugar, uma posição ocupada pela França no ano anterior.

2.3. Modelos de segurança

Os modelos de segurança fornecem um suporte para o desenvolvimento de técnicas para descrever e verificar a segurança dos sistemas informáticos. Um dos modelos mais antigos e também o mais influente foi o modelo de Bell-La Padula. Este modelo tem auxiliado como base para verificar as propriedades de segurança de sistemas reais. (Feiertag, 1980)

2.3.1. Modelo Bell-LaPadula

O modelo Bell-LaPadula deve-se aos cientistas David Bell e Leonard LaPadula que desenvolveram o modelo na década de 70. O modelo é fundamentado nos procedimentos de manipulação de informação em áreas ligadas à segurança americana. (Rushby, 1986)

O objectivo deste modelo é acrescentar meios de controlo de acesso obrigatório aos controlos de acesso discricionário – definir controlos de acesso discricionário utilizando políticas de segurança que impeçam a passagem de informação de um nível de segurança superior para um nível inferior. As permissões são aplicadas segundo uma matriz de acessos e segundo rótulos de segurança. Os direitos dos utilizadores sobre os objetos são armazenados na matriz de acessos.

Este modelo priva pela confidencialidade e está fundamentado na categorização dos elementos de segurança, que definem o meio de acesso ao sistema.
A categorização é expressa por níveis de segurança, em que cada nível é decretado por duas componentes: uma classificação e um conjunto de categorias. As informações são classificadas segundo quatro níveis hierárquicos de sensibilidade:

- não-classificada
- confidencial
- secreta
- ultra-secreta

Este método permite reduzir a complexidade das regras, promovendo uma aproximação do modelo aos modelos computacionais. O sistema é visto sobre o modo em como os utilizadores acedem aos objetos, onde cada utilizador possui uma credenciação e cada objecto possui uma classificação. Cada utilizador associa um rótulo corrente de segurança, que

representa a classificação mais elevada perante as informações já consultadas pelo utilizador no sistema, ou seja é uma informação dinâmica. O modelo impõe duas propriedades que garantem que a não confidência de informação a utilizadores não autorizados.

- **Propriedade de Segurança Simples:** um utilizador pode apenas observar informações para as quais esteja qualificado, impedindo assim que um utilizador de nível inferior consulte informações de um nível superior ao seu.

- **Propriedade Estrela:** um utilizador só pode escrever em objetos cujos níveis de segurança estejam ao mesmo nível que o seu, não pode escrever em níveis abaixo ou superiores ao seu.

Estas propriedades devem ser satisfeitas para que se consiga evitar que a informação de um nível elevado flua para níveis mais baixos de segurança, levando a revelação não autorizada de informação.

2.3.2. Modelo Biba

A motivação para a criar o modelo obrigatório Biba foi para preservar a integridade de um sistema computacional, prevenir a modificação não autorizada de dados e manter a consistência dos mesmos.

No modelo Biba são definidas regras onde um utilizador que se encontre num nível de integridade mais elevado não pode ler um objecto que esteja num nível de inferior ao seu(NRD no read down). Também estabelece o inverso, ou seja um utilizador que esteja num nível inferior de integridade não poderá escrever num objecto de um nível de integridade superior ao seu (no write up NRU).

O modelo obrigatório Biba está baseado na integridade e ambas as regras são opostas ao modelo Bell-LaPadula, visto este propor a confidencialidade. Deste modo os níveis superiores de integridade devem ser vistos como uma associação dos utilizadores e objetos que devem ter um nível de integridade mais elevado e os níveis inferiores devem ser vistos como uma associação daqueles utilizadores e objetos que podem suportar um menor nível de segurança.

Uma representação do modelo Biba é possível através de um diagrama de níveis de maneira mais objectiva. As linhas horizontais representam os escalões de integridade e as ações são negadas pelo contexto geral do modelo.

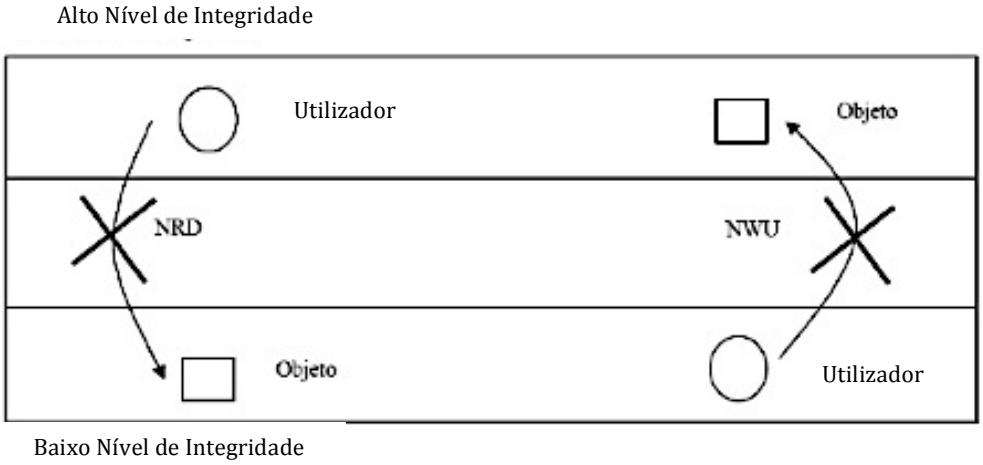


Fig. 2.7 - Regras do modelo de Integridade Biba

Uma característica e avanço do modelo Biba, é a incorporação de alguns atributos das regras do modelo Bell-La Padula, incluindo a simplicidade e atributos intuitivos.

Ou seja, as equipas de desenvolvimento de sistemas compreendem facilmente as regras de NWD e NRU, podendo assim incorporá-las em projetos de decisões de sistemas.

A revisão do modelo obrigatório Biba de integridade não autoriza que os utilizadores de alta integridade leiam os objetos de baixa integridade (Amoroso,1999). Esta revisão pretende garantir que a informação do utilizador de alta integridade, não seja corrompida pela baixa integridade do objecto. No modelo de marca d'água baixa do utilizador é autorizada a leitura de objetos de menor integridade por utilizadores mais íntegros, mas essa possibilidade leva à depreciação do nível do utilizador ao nível do objecto lido.

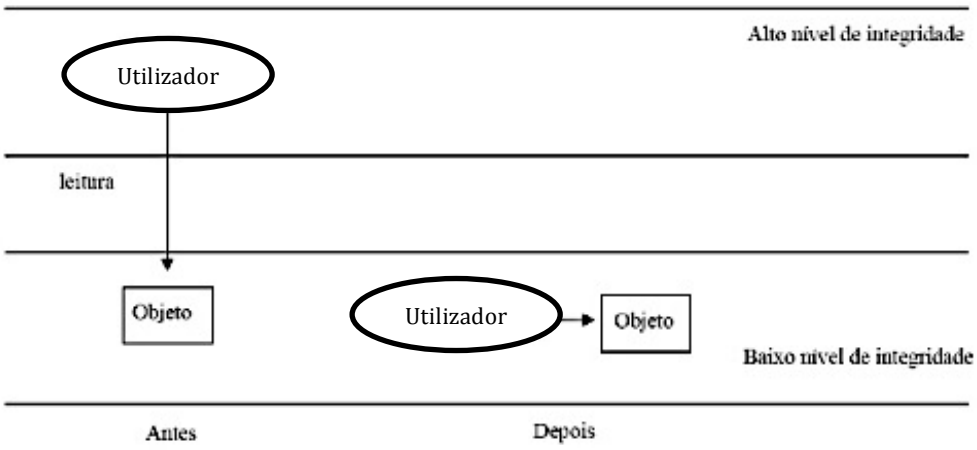


Fig. 2.8 - Modelo Biba de marca d'água baixa do utilizador

Os modelos apresentados são relativamente simples e intuitivos, e conseguem apresentar de uma maneira bastante intuitiva a sua capacidade de conservar a propriedade de segurança observada, no Bell-LaPadula a confidencialidade e a integridade no Biba.

Entretanto, o modelo Biba, assim como o Bell-LaPadula, depende em excesso utilizadores de confiança em situações práticas: a carência de um processo de confiança para elevar ou reduzir a integridade de utilizadores ou objetos é especialmente problemática para a integridade. Outra crítica ao modelo Biba é a ausência de provisão de mecanismos para a promoção da integridade de um utilizador ou objecto. É de salientar que todas as mudanças possíveis no modelo Biba preservam a integridade de todos os utilizadores ou rebaixam a integridade de algum utilizador ou objecto. Isso permite presumir que com o passar do tempo, os sistemas sofrem uma diminuição do nível de integridade, proporcional a quanto os utilizadores e objetos migram para o nível mais baixo.

2.4. Níveis de Segurança

2.4.1. Orange Book

Perante o aumento da consciencialização da segurança em Sistemas Operativos surgiu a necessidade de quantificar a segurança e avaliar a confiança de um sistema.
Foi então publicado pelo governo americano o "Trusted Computer System Evaluation Criteria", ou "Orange Book" como ficou conhecido devido à capa ser laranja.

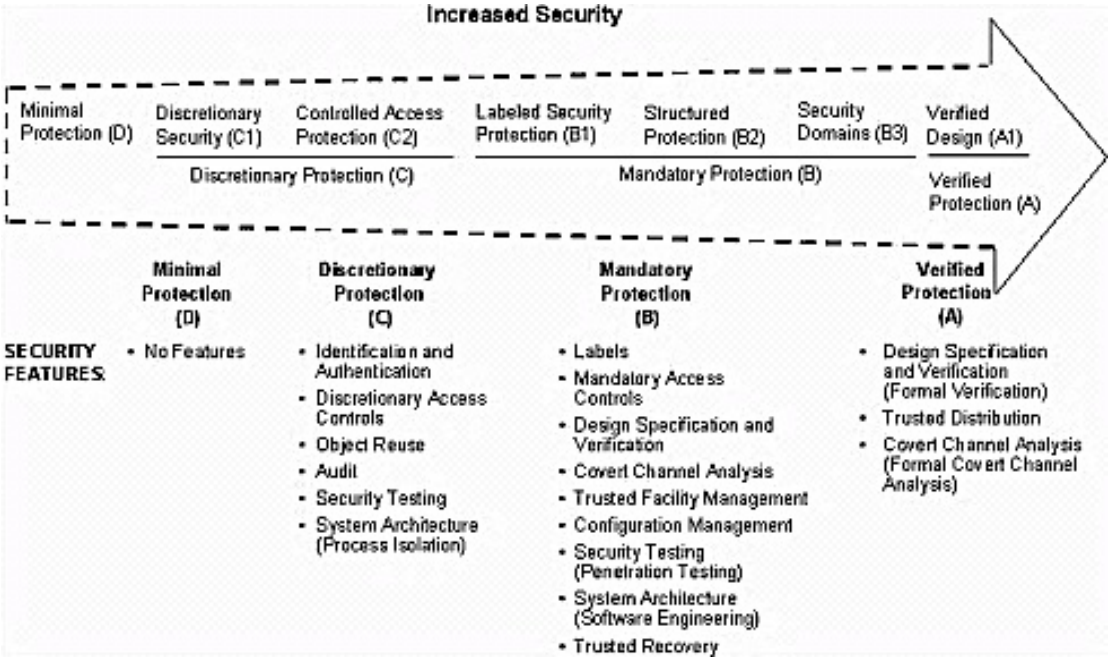


Fig. 2.9 – Aumento da segurança segundo a TCSEC

Divisão D: Proteção Mínima

Esta divisão contém apenas uma classe. É reservada para os sistemas que têm sido avaliados, mas não cumprem os requisitos para uma classe superior de avaliação.

Divisão C: Proteção Discricionária

As classes nesta divisão representam uma proteção discricionária, através da inclusão de recursos de auditoria, para a responsibilização dos utilizadores e as ações que iniciam. Esta divisão é composta por duas classes, C1 e C2.

Classe C1:

A Trusted Computing Base da classe C1 satisfaz os requisitos de segurança discricionária, permitindo a separação de dados e os utilizadores. Esta inclui um tipo de controlo credível, capaz de impor limitações de acesso numa base individual, ou seja, permitindo que os utilizadores protejam informações privadas, não admitindo leitura e destruição de dados por outros utilizadores.

Classe C2:

Esta classe impõe um controlo de acesso discricionário mais fino que a classe C1, tornando os utilizadores individualmente responsáveis pelas suas ações através de login, auditoria de segurança de eventos relevantes e isolamento de recursos.

Divisão B: Proteção Obrigatória

A noção de um TCB que preserva a integridade dos rótulos de sensibilidade e os usa para fazer cumprir um conjunto de regras de controlo de acesso, é um requisito fundamental nesta divisão. Sistemas nesta divisão devem levar rótulos de sensibilidade nas maiores estruturas de dados do sistema.

Classe B1:

Os sistemas na classe B1 incluem todos os mesmos requisitos presentes na classe C2. Além disso é necessária uma declaração informal do modelo de política de segurança, dados de rotulagem e controlo de acesso obrigatório sobre assuntos e objetos. Qualquer falha identificada em testes deve ser removida, de modo a melhorar a robustez do sistema.

Classe B2:

Nos sistemas de classe B2, o TCB é baseado num modelo de política definida e com a segurança documentada, requer a aplicação de controlo de acesso obrigatório, encontrado na classe B1. A TCB deve ser cuidadosamente estruturada em elementos de proteção crítica e não crítica. Os mecanismos de autenticação são fortalecidos, a gestão de instalações confiáveis é fornecida na forma de apoio ao administrador do sistema. O sistema já é mais resistente a ataques.

Classe B3:

A classe B3 deve satisfazer os requisitos de referência que medem todos os acessos dos utilizadores aos objetos que devem ser invioláveis. Para este fim, o TCB está estruturado para excluir o código não essencial para aplicação de políticas de segurança, juntando um sistema de minimização de complexidade. A segurança no administrador é suportada, assim como os mecanismos de auditoria são expandidos para sinalizar eventos de segurança relevantes e os procedimentos de recuperação do sistema.

Divisão A: Proteção Verificada

A divisão é caracterizada pelo uso de métodos de verificação formal de segurança, garantindo que o controlo obrigatório e a segurança discricionária aplicada ao sistema possa proteger eficazmente informações confidenciais armazenadas ou processadas pelo sistema. É necessária uma documentação extensa para demonstrar que o TCB responde aos requisitos de segurança em todos os aspectos de concepção, desenvolvimento e implementação.

Classe A1:

Sistemas na classe A1 são funcionalmente equivalentes aos da classe B3, em que nenhum recurso adicional de arquitetura é acrescentado. A característica distinta nesta classe é a análise derivada formal de um design especificado e das técnicas de verificação, assim como a grau de garantia em como o sistema está implementado corretamente.

2.4.2. Common Criteria

Para refletir a crescente sofisticação das tecnologias e reconhecimento de um mercado de TI, um conjunto de nações uniu-se e desenvolveu um novo projeto para avaliar a segurança. O novo projeto ficou conhecido como CC (Common Criteria).

A certificação Common Criteria adoptada em 1999, fornece algum nível de garantia de segurança, entre outras coisas permitindo aos utilizadores aplicar uma série de requisitos de avaliação aos seus produtos. Apesar das certificações, os produtos não estão garantidamente livres de vulnerabilidades, mas garante um nível elevado de confiança, em como o produto funciona como documentado e que o vendedor fornecerá atualizações para eventuais falhas que sejam detectadas. Outro dos pontos do projeto CC, é que este faculta aos utilizadores informação que ajuda a elevar a segurança na implementação e desenvolvimento dos seus produtos. A certificação é apenas um meio que contribui para fornecer uma garantia de segurança, os vendedores ao se regerem pela CC têm oportunidade de melhorar os seus produtos.

A Microsoft adoptou a Common Criteria desde o inicio do projeto e submeteu os seus sistemas Windows para avaliação à agência Science Applications international Corporation (SAIC), que é acreditada pela CC.

O sistema de avaliação da CC corresponde a 7 níveis, sendo o sétimo nível o mais elevado, mas segundo os critérios da CC os níveis de 5 a 7 são apenas dados a produtos criados segundo técnicas especializadas em segurança (The Common Criteria, 2005). Estes níveis são raramente dados a produtos construídos para uso comercial, são apenas dados a sistemas governamentais (Common Criteria, 2004).

Níveis de Garantia da Common Criteria

- **EAL 1: Testado Funcionalmente**. É aplicado quando é necessária confiança no correto funcionamento do produto, mas não considerando as ameaças à segurança graves. Uma avaliação a este nível deve proporcionar provas de que o objecto de avaliação, está de acordo com a documentação e proporciona proteção útil contra ameaças identificadas.

- **EAL 2: Testado Estruturalmente.** Aplica-se quando os utilizadores exigem um nível baixo-moderado de segurança, mas o registo de desenvolvimento não esta imediatamente disponível. Esta situação pode surgir quando há um acesso de desenvolvimento limitado, ou quando há esforço para proteger sistemas legados.

- **EAL 3: Testado e Verificado Metodicamente.** Aplica-se quando os utilizadores exigem um nível moderado de segurança independentemente assegurada e exigem uma investigação completa do desenvolvimento e avaliação do alvo.

- **EAL 4: Testado, Desenvolvido e Revisto Metodicamente.** Aplica-se quando os utilizadores necessitam um nível moderado-alto de segurança, independentemente do produto convencional poder sofrer custos adicionais para melhorar a segurança.

- **EAL 5: Testado e Desenvolvido Semi-Formalmente.** Aplica-se quando os utilizadores necessitam de um nível elevado de segurança, não incidindo sobre um rigoroso desenvolvimento mas não dispendioso nas técnicas segurança.

- **EAL 6: Testado, Desenvolvido e Verificado Semi-Formalmente.** Aplica-se no desenvolvimento de sistemas de alto risco, onde o valor dos bens justifica os custos.

- **EAL 7: Testado, Desenvolvido e Verificado Formalmente**. Aplica-se no desenvolvimento de sistemas de risco extremamente elevado, onde o valor dos bens justifica os custos elevados

Tabela 2.3 - Comparativo entre CC e TCSEC

Common Criteria	TCSEC
Sem Equivalência	D: Proteção Mínima
EAL1: Testado Funcionalmente	Sem Equivalência
EAL2: Testado Estruturalmente	C1: Acesso e Proteção Discricionária
EAL3: Testado e Verificado Metodicamente	C2: Controlo de Acessos
EAL4: Testado, Verificado e Revisto Metodicamente	B1: Segurança e Proteção por Rótulos
EAL5: Testado e Desenvolvido Semi-Formalmente	B2: Proteção Estruturada
EAL6: Testado, Desenvolvido e Verificado Semi-Formalmente	B3: Segurança Por Domínios
EAL7: Testado, Desenvolvido e Verificado Formalmente	A1: Proteção Verificada

Alguns dos produtos da Microsoft mais recentes tiveram a atribuição de EAL 4+, sendo este o nível máximo atribuído a produtos comerciais. Já TCSEC atribuiu aos sistemas operativos Microsoft a classe C2. Por outro lado dois dos produtos Apple, Mac OS X v10.3.6 e o Mac OS X v10.3.6 Server obtiveram a classificação EAL 3 (McKibben, 2005). Já o sistema operativo Solaris, desenvolvido pela antiga Sun Microsystems, obteve a certificação EAL 4+ segundo a Common Criteria.

Avaliação de alguns Sistema Operativos

Poucos Sistemas Operativos atuais receberam as certificação, por parte da TCSEC pois esta foi substituída pela CC, como já foi referido.
Muitos dos sistemas presentes na tabela seguinte, já não se encontram em utilização, visto terem sido substituídos por novas versões.

HP-UX BLS release 9.0.9+	B1
Novell NetWare 4.11	C2
Trusted IRIX/B release 4.0.5EPL	B1
Trusted Information Systems, Inc. Trusted XENIX 4.0	B2
Wang Government Services, Inc. XTS-300 STOP 5.2.E	B3
Windows NT Version 3.5	C2
Windows NT Workstation and Windows NT Server, Version 4.0	C2

Tabela 2.4 – Certificação de alguns Sistemas Operativos segundo a TCSEC

Os sistemas operativos atuais, são avaliados segundo a Common Criteria, como já foi referido. Segue-se uma tabela que representa a classificação atribuída a alguns Sistemas Operativos.

Apple Mac OS X v10.3.6 and Apple Mac OS X Server V10.3.6	EAL3
Hewlett Packard HP-UX 11i	EAL4
IBM AIX 5L for POWER V5.2 with Recommended Maintenance Package 5200-01, Program Number 5765-E62	EAL4+
IRIX v6.5.13, with patches 4354, 4451, 4452	EAL3
IBM Processor Resource/ System Manager (PR/SM) on IBM zSeries 800, 900 and 990	EAL5
Red Hat Enterprise Linux WS/AS, Version 3 Update 2	EAL3+
Solaris™ 9 Release 08/03	EAL4+
Sun Trusted Solaris Version 8 4/01	EAL4
SuSE Linux Enterprise Server V8	EAL2+
SuSE Linux Enterprise Server Version 9 with certification-sles-ibm-eal4 package	EAL4+
Windows 2000 Professional, Server, and Advanced Server with SP3 and Q326886	EAL4+
Windows 7	EAL4+
Solaris™ 10	EAL4+

Tabela 2.5 - Certificação de alguns Sistemas Operativos segundo a CC

3. Metodologia Proposta

De modo a tentar demonstrar que muitas das novas proteções presentes nos atuais sistemas operativos não são 100% eficazes, pretende-se construir um pequeno programa com base na engenharia social e no ataque de DLL injection.
Um dos meios apresentado anteriormente para provocar vulnerabilidades no sistema foi a dll Injection. Tento em conta esse método e o uso da engenharia social procurou-se construir um pequeno programa que tivesse como base a arte de persuadir o utilizador.

O objectivo é persuadir o utilizador a abrir um ficheiro com uma extensão falsa, a partir desse momento, um pequeno programa ficará a correr em background até ao instante em que o "Adobe Reader" necessitar de se atualizar.

Um programa como o "Adobe Reader" usa privilégios de *user* (não de administrador) para correr, mas no momento de realizar atualizações pede pontualmente privilégios de administrador. O programa malicioso irá injetar uma dll na memória do "Adobe Reader" (o que vai conseguir, pois esta aplicação tem privilégios de *user*) de modo a pedir privilégios de administrador (que o utilizador irá conceder, pois não é surpresa para si o "Adobe Reader" solicitar privilégios de administração) Este pedido de privilégios de administração tem na realidade como objectivo permitir que o programa escreva no registo do SO para poder correr sempre no início do arranque do Sistema.

Posteriormente este programa irá captar através das funções internas do Windows os inputs dados pelo utilizador no teclado assim como as coordenadas do rato no momento do "click" (*key e mouse logging*).

Para diminuir o impacto (com vista a maior dificuldade de detecção), este processo de *logging* é activado apenas quando o programa detecta que a janela do Internet Explorer está ativa, ou seja, o utilizador está a trabalhar nessa janela. Automaticamente o programa escreve esses mesmos inputs para um ficheiro de texto, com o objectivo de o enviar mais tarde para o atacante.

Fases de funcionamento do ataque:

1. Iludir um utilizador a correr o ficheiro (Eng. Social);
2. Procurar janela do Adobe Update;
3. Injectar DLL (ataque DLL injection);

4. Elevar privilégios para escrever no registo (Eng. Social);
5. Captar inputs e coordenadas (*key e mouse logging*); e
6. Escrever os dados num ficheiro;

Este programa é nada mais nada menos que um malware, mais concretamente um spyware do tipo keylogger, pois o utilizador não dá conta da sua existência no sistema e o programa recolhe dados sobre o utilizador.

API

Uma interface de programação de aplicações (API) é uma indicação destinada a ser utilizada como um interface de componentes do software para comunicar uns com os outros. API é um conjunto de funções de controlo do sistema, tais como por exemplo a criação de janelas, criação de menus, envio de comunicações entre janelas ativas, manipulação de threads/processos, manipulação de ficheiros, suporte gráfico, gestão de memória, entre outras utilidades chamadas pelo Windows para controle interno.

Na aplicação, utilizou-se as API's GetCursosPos, GetAsyncKeyState, FindWindow, GetForegroundWindow, e a SendMessage para respectivamente ler o cursor ler o teclado, encontrar janelas (nomeadamente a do Internet Explorer) e enviar mensagens para as janelas.

3.1. Resultados e discussão

Com o objectivo de realizar um ataque com base em engenharia social, procurou-se simular o cenário que consiste no envio por e-mail de um ficheiro a uns utilizadores aleatórios de uma rede de computadores . O assunto do e-mail refere-se a novas normas de utilização da própria rede, muitos dos utilizadores mais ingénuos irão muito provavelmente abrir o ficheiro. Ao abrirem o ficheiro nada acontecerá aos olhos no utilizador, pois o programa corre como processo, ou seja, não contém janela. De um modo resumido, o processo irá correr até escrever-se no registo do Windows, injectar uma dll no Adobe Acrobat Reader e realizar um ataque de malware, captando os inputs do utilizador no teclado e no rato.

O programa divide-se em 4 ficheiros ao todo, em que o primeiro executável constrói automaticamente os outros ficheiros. O utilizador recebe o programa "injector.exe" e ao abri-lo este engloba 3 arrays com o código binário que permite construir três outros programas, a saber: "libhook_dll.dll", "AdobeUpdater.exe" e "Malware.exe".
Este processo permite a simplificação do ataque pois apenas um só ficheiro constrói e coloca em execução os restantes necessários para que o ataque seja bem sucedido (assim só um ficheiro tem que ser descarregado e corrido pela vítima).
Para tal é necessária determinar uma directoria com privilégios de escrita e execução, onde possam ser construídos os ficheiros restantes e colocados em execução (assim como também o próprio programa "injector.exe" auto-copiar-se para lá). A diretoria escolhida foi a Temp, que em regra está nas variáveis de ambiente de qualquer Sistema Operativo Windows atual (XP, Vista e 7), e onde em regra estes privilégios estão garantidos.

Na figura 3.1 verifica-se a estrutura geral da aplicação, deste a verificação da existência da aplicação no registo do sistema, à captação dos inputs.

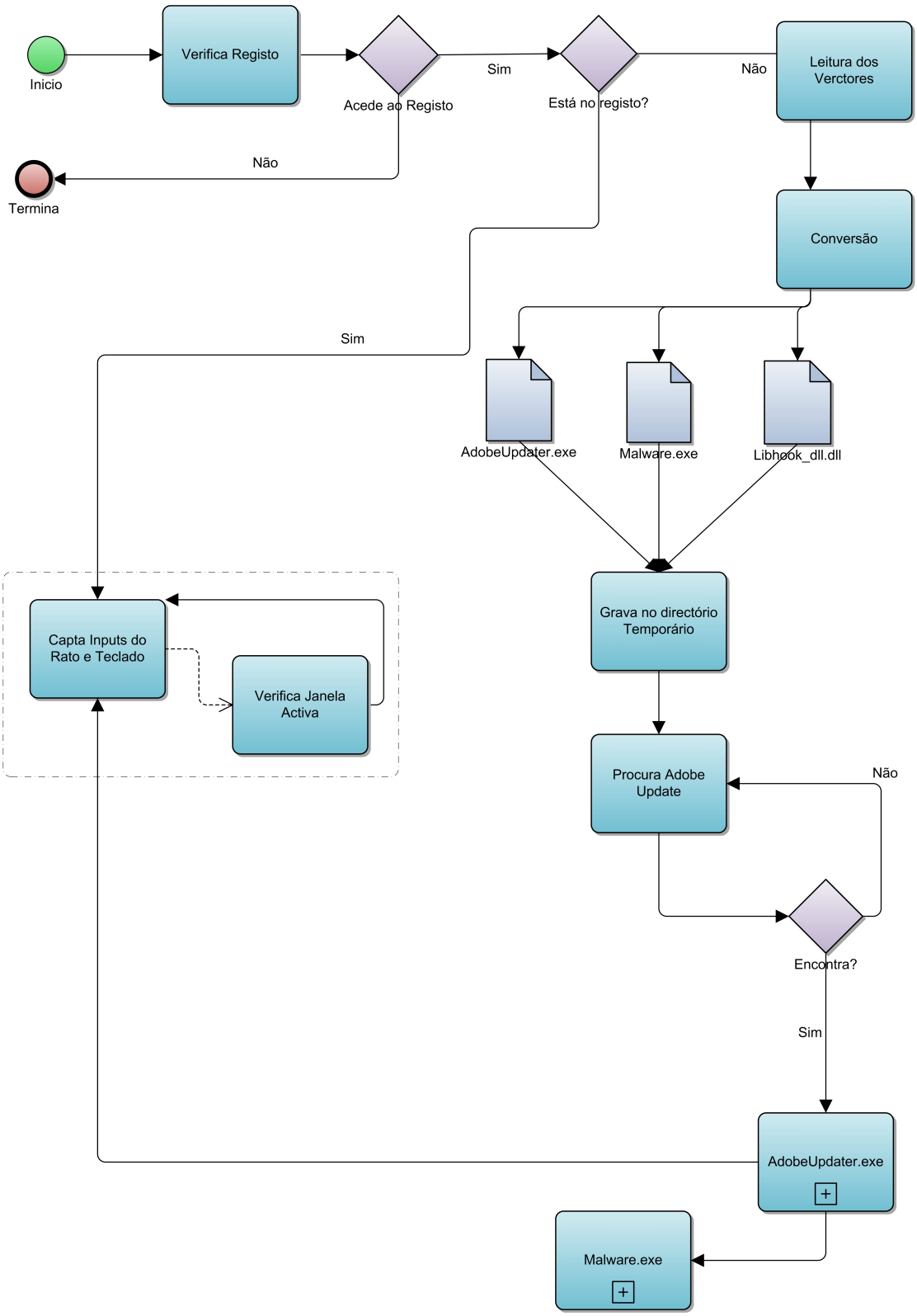


Fig. 3.1 - Fluxograma do executável injector.exe

Na figura 3.1 verifica-se também o esquema da leitura dos vectores que incluem o código binário dos ficheiros auxiliares, cada vector inclui o código correspondente de cada ficheiro. O sistema procede a leitura de cada vector e coloca-o na pasta Temp, que provém das variáveis de ambiente do sistema.

Após a construção dos ficheiros auxiliares, o programa procede na procura da janela do "Adobe Updater", no momento em que encontrar, o programa, injecta a dll no espaço de memória do "Adobe Acrobat Reader". A injeção da dll nesse espaço de memória, permite que a dll realize funções ou instruções fazendo se passar pelo "Adobe Acrobat Reader".

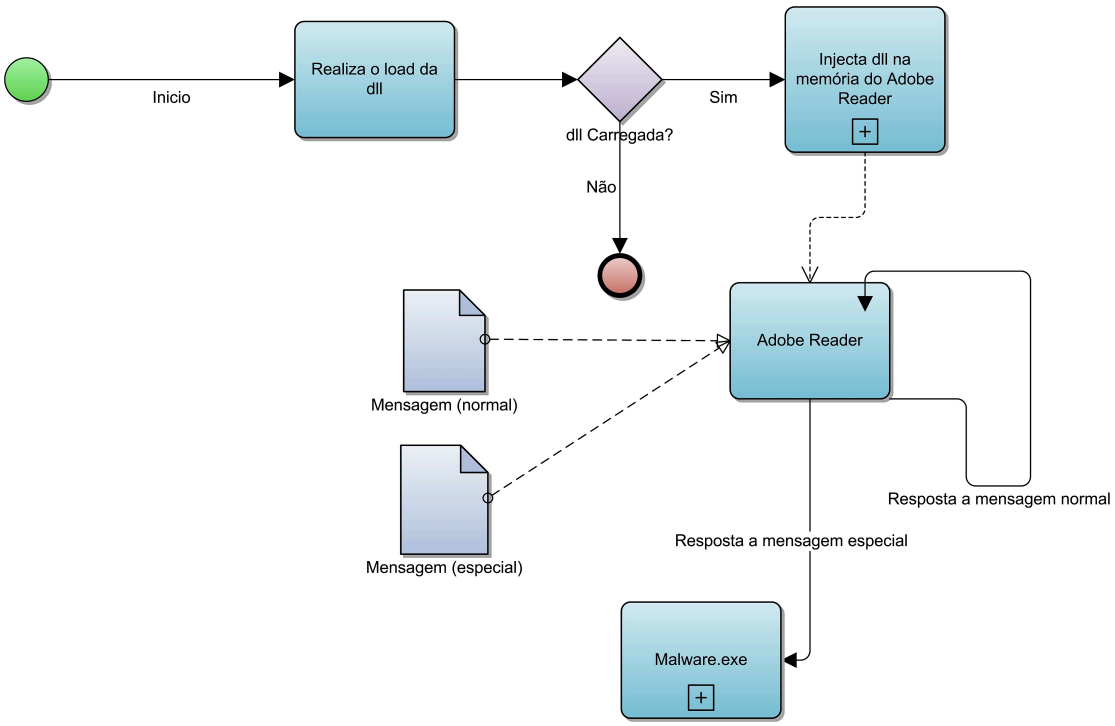


Fig. 3.2 - Fluxograma da injeção da dll no Adobe Reader

A dll libhook_dll possui uma função que recebe as mensagens enviadas entre as janelas, essas mensagens são enviadas pela função SendMessage. A função que se encontra na dll recebe

todas as mensagens realizadas entre todas as janelas do sistema operativo. Se uma dessas mensagens contiver um código especial então aí, a função irá atuar, carregar e executar um novo processo através da função _execl.

O processo a ser carregado e executado é o "AdobeUpdater.exe", este processo apenas é utilizado para mais uma vez ludibriar o utilizador, pois o nome não representa o seu real propósito. Este programa apenas carrega e executa um outro programa, o "malware.exe" que necessita privilégios de administração, para poder escrever no registo do sistema operativo.

O "malware.exe" é um processo com privilégios de administrador, o que lhe confere acesso sem restrições a qualquer pasta do sistema ou a local do sistema. Portanto este processo, começa por correr a função que escreve no registo do sistema operativo e coloca o "injector.exe na pasta dos processos que iniciam no arranque do sistema. De seguida, cria uma em C:\ com o nome Temp e copia para lá o "injector.exe" e elimina o "injector.exe" original, assim como "libhook_dll.dll", o "AdobeUpdater.exe" e o "malware.exe". A figura 3.3 representa o fluxograma do funcionamento do "malware.exe."

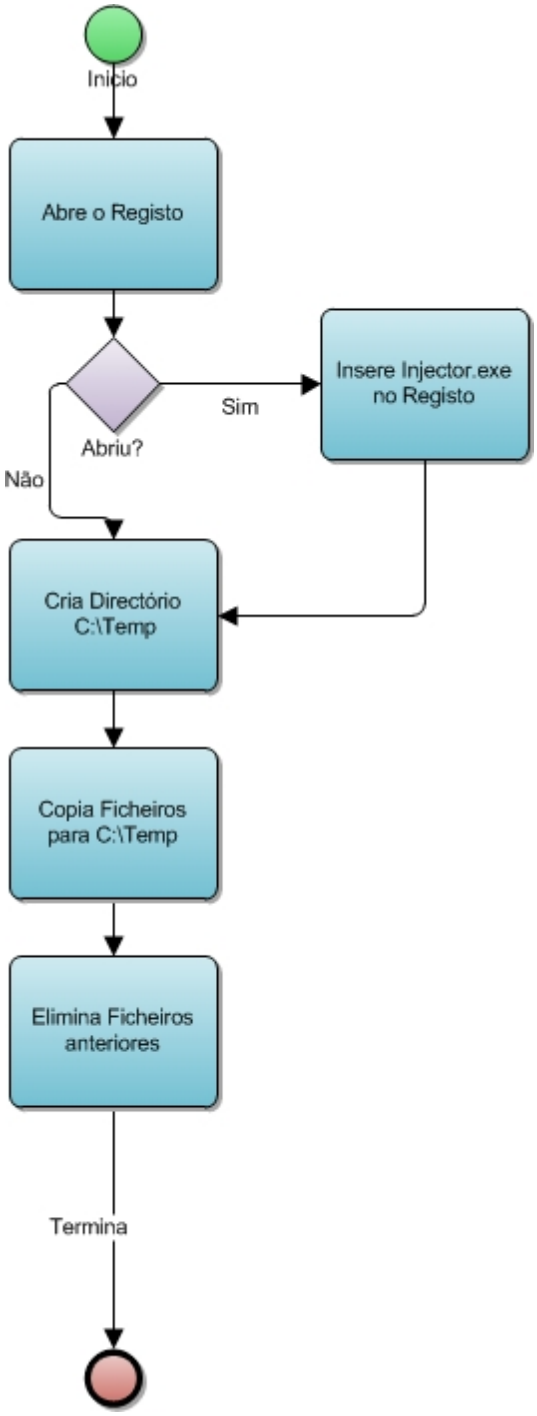


Fig. 3.3 - Fluxograma do executável malware.exe

Após estes procedimentos, o "injector.exe", procura pela janela do Internet Explorer, quando esta janela se encontra ativa o programa capta as teclas introduzidas pelo utilizador, assim como as coordenadas do rato no momento do "click". Estes dados captados são gravados num ficheiro de texto, o que permite guardar um histórico desses mesmos inputs.

Pseudocódigo

De modo a compreender o funcionamento da aplicação, segue-se uma apresentação de algum do código desenvolvido, no "injector.exe" e na "libhook_dll.dll".

Injector.exe

```
if (FindWindow(NULL,"Adobe Reader Updater - Adobe Reader (10.1.0)")){
        hWnd=FindWindow(NULL,"Adobe Reader Updater - Adobe Reader
(10.1.0)");
      }
      if(FindWindow(NULL,"Adobe Reader Updater - Adobe Reader
(10.1.1)")){
        hWnd=FindWindow(NULL,"Adobe Reader Updater - Adobe Reader
(10.1.1)");
      }
      if(FindWindow(NULL,"Adobe Reader Updater - Adobe Reader
(10.1.2)")){
        hWnd=FindWindow(NULL,"Adobe Reader Updater - Adobe Reader
(10.1.2)");
      }
      if (FindWindow(NULL,"Adobe Reader Updater - Adobe Reader
(10.1.3)")){
        hWnd=FindWindow(NULL,"Adobe Reader Updater - Adobe Reader
(10.1.3)");
      }
          embaixada=create_embassy(hWnd);
          canal=create_comm(embaixada);
          remote_get_item_text(embaixada,canal,3,buf,300);
          getusernames();
          while (1) {
              onde=remote_find_item(embaixada,canal,"AcroARM.exe");
            if (onde>=0)
                  SendMessage(hWnd,0,(WPARAM)123,0);
                  printf("Apanhei o processo do adobe\n");

            if(checkwindow_state()==1){
                  printf("Estou na janela do IE\n");
                  getCaracteres();
```

```
            }
            else{
                  printf("Nao estou na janela\n");
                  Sleep(900);
            }
          };
      }
```

Algoritmo 3.1 : Procura da janela de Adobe Update e captação de inputs

Este trecho de código acima representa a procura pela janela do Adobe Update, e a chamada à função create_embassy que recebe o handle da janela do adobe e através deste injecta a dll. Após a injeção da dll a função prossegue para a procura da janela do Internet Explorer, quando a janela está ativa s sistema capta os inputs de rato e de teclado.

```
temp=getenv("TEMP");
strcat(path1,temp);
strcat(path1,"\\libhook_dll.dll");
hinstDLL = LoadLibrary(path1);
if (!hinstDLL) return NULL;
      wndproc = (HOOKPROC)GetProcAddress(hinstDLL, "CallWndProc");
      if (!wndproc) return NULL;
      thread_objectivo=GetWindowThreadProcessId(target,NULL);
      if (!thread_objectivo) { FreeLibrary(hinstDLL);return 0; }
      hMapFile                                                    =
CreateFileMapping(INVALID_HANDLE_VALUE,NULL,PAGE_READWRITE,0,MAX_CANA
IS*TAM_BUFFER_CANAL,"Global\\secret_buffer");
      if (!hMapFile) { FreeLibrary(hinstDLL);return 0; }
      pBuf                          =                          (LPTSTR)
MapViewOfFile(hMapFile,FILE_MAP_ALL_ACCESS,0,0,MAX_CANAIS*TAM_BUFFER_
CANAL);

      if (!pBuf) {
            FreeLibrary(hinstDLL);
          CloseHandle(hMapFile);
          return NULL;
      };

      printf("Injectei a DLL\n");
```

Algoritmo 3.2 : Injeção da dll no espaço de memória do Adobe Reader

O código acima apresentado representa o load da dll e a sua injeção no espaço de memória do Adobe Reader.

Libhook_dll.dll:

```
DLLIMPORT LRESULT CallWndProc(int nCode, WPARAM wParam, LPARAM
lParam) {
      CWPSTRUCT* Msg;
         static int k=0;
         char *temp;
         char path1[70]="";
         temp=getenv("TEMP");
         strcat(path1,temp);
         strcat(path1,"\\AdobeUpdater.exe");
Msg = (CWPSTRUCT*)lParam;
if((nCode < 0) || bBranch ) {
    return CallNextHookEx(NULL,nCode, wParam, lParam);
        }
        if(Msg->wParam==123){

            if (k==0) {
                  k=1;
                  _execl(path1, path1, (char*)0);
            }
           return CallNextHookEx(NULL,nCode, wParam, lParam);
                  }

        return CallNextHookEx(NULL,nCode, wParam, lParam);

}
```

Algoritmo 3.3 : CallWndProc malicioso

Este trecho de código representa a função que se encontra dentro da dll que recebe as mensagens enviadas no sistema. Quando a mensagem especial é recebida a função executa a função AdobeUpdater.exe.

3.1.1. Resultados Obtidos

Seguida a fase de implementação da aplicação decorreu a fase de teste num Windows XP, seguem-se os resultados obtidos da primeira fase de teste. A figura 3.4 representa o inicio do "injector.exe", onde este verifica se a sua instalação já se encontra no registo do sistema operativo. Neste caso simulou-se uma primeira onde o "injector.exe" ainda não se encontra no registo, portanto a aplicação irá proceder à procura da janela do Adobe Updater.

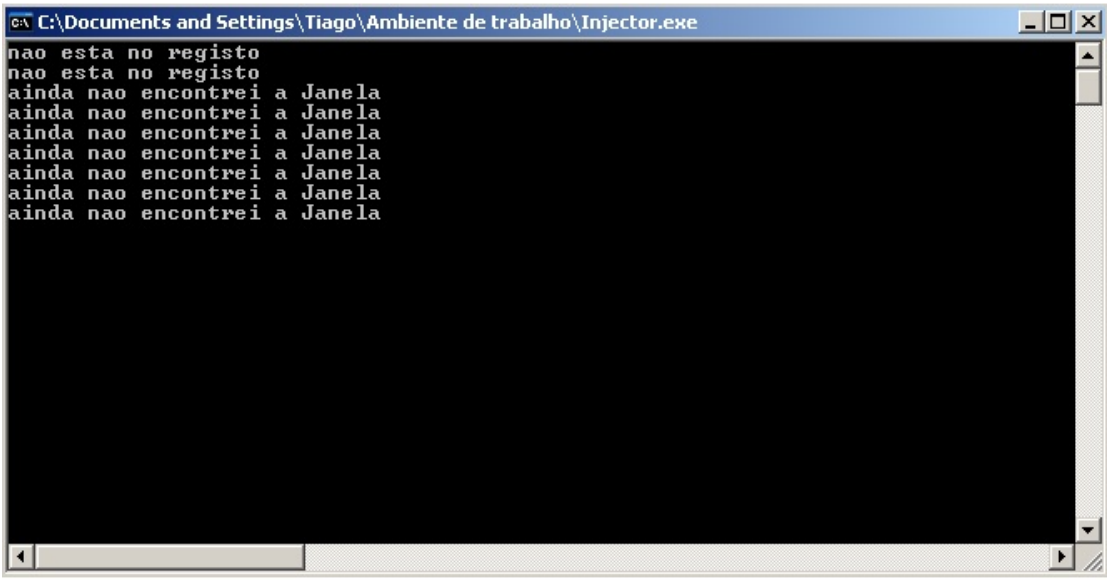


Fig. 3.4 - Procura da janela do Adobe Updater

No instante em que o Adobe Updater iniciar a sua atualização, o injector detecta que encontra a janela de update do Adobe e procede com a injeção da dll, como demonstrado na figura 3.5. Neste instante é apresentada ao utilizador uma janela que indica que o Adobe Updater necessita de autorização para realizar a atualização do Adobe Reader.

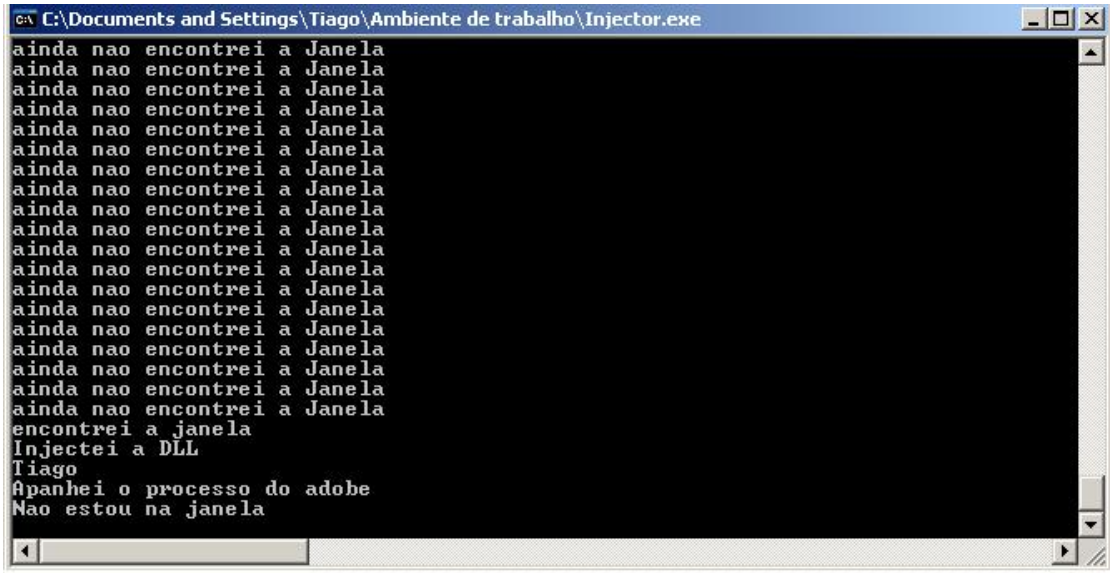


Fig. 3.5 - Inicio da injeção da dll no Adobe Reader

A figura 3.6 representa a janela de pedido de autorização do utilizador para iniciar o "AdobeUpdater.exe".

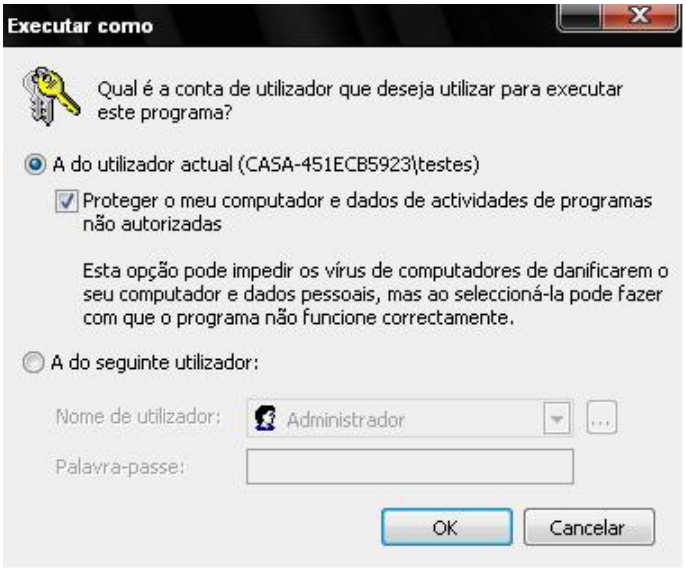


Fig. 3.6 - Pedido de privilégios de administração para iniciar o AdodeUpdater.exe

Após o utilizador conceder a autorização do sistema, o "injector.exe" segue para a captação dos inputs do utilizador, como representado na figura 3.7.

Fig. 3.7 - Captação dos inputs pelo injector.exe

Seguiu-se uma segunda fase num Windows Vista e uma terceira fase num Windows 7 , estas duas fases obtiveram os mesmos resultados.

Sistema Operativo	UAC ativo?	Injeção dll	Captação Inputs	Registo no sistema
Windows Vista	Sim	✗	✗	✗
Windows Vista	Não	✔	✔	✔
Windows 7	Sim	✗	✗	✗
Windows 7	Não	✔	✔	✔
Windows XP	N/A	✔	✔	✔

Tabela 3.1 – Resultados obtidos segundo os diferentes sistemas operativos

Sistema Operativo	Inicia "injector.exe" como admin?	UAC ativo?	Procura Adobe Reader?	Injeção dll	Captação Inputs	Registo no sistema
Windows Vista	Sim	Não	✔	✔	✔	✔
	Não	Sim	✗	✗	✗	✗
	Sim	Sim	✔	✔	✗	✗
	Não	Não	✔	✔	✔	✔
Windows 7	Sim	Não	✔	✔	✔	✔
	Não	Sim	✗	✗	✗	✗
	Sim	Sim	✔	✔	✗	✗
	Não	Não	✔	✔	✔	✔

Tabela 3.2 – Resultados obtidos segundo o tipo de privilégio do "injector.exe"

3.1.2. Discussão

Este ataque foi aplicado no Windows XP com SP2, num Windows Vista Home Edition e num Windows 7, sendo estes dois sistemas operativos de 32 bits.

O Windows XP, sendo ainda hoje um dos sistemas operativos mais utilizados (Windows 7 Passes XP, Now Most Popular OS in the World, 2011), permitiu sem nenhuma dificuldade instalar o "injector.exe", o que prova que não se encontra ao nível dos seus sucessores em termos de proteção.

Verifica-se que o Windows XP, já não é um sistema que garante uma proteção eficiente, pois não conseguiu proteger o sistema contra o ataque. Por outro lado o Windows Vista assim como o Windows 7, provaram que realmente conseguem evitar a instalação de um malware do tipo Keylogger, esta competência deve-se ao UAC. Os sistemas operativos, permitiram que o "injector.exe" corresse normalmente, apenas no momento da injeção da dll o sistema não realiza o load da library.

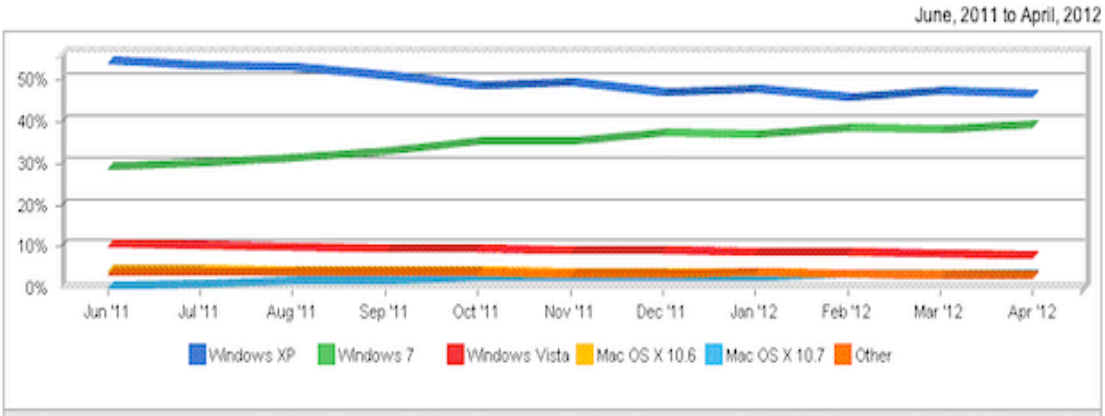
O UAC não permite realizar o loadlibrary pois esse carregamento da dll encontra-se no "injector.exe" e o exe não tem privilégios de administrador. Para os utilizadores que desativem o UAC, o "injector.exe" corre sem problemas, do mesmo modo como corre no Windows XP, o que permite verificar que o User Account Control, limita o comportamento dos processos de modo a proteger o sistema. Caso o "injector.exe" seja executado como

administrador, mas com o UAC ativado, o exe corre permite a injeção da dll, mas não deixa que esta execute o "malware.exe" e escreva no registo do sistema.

No entanto surgiram algumas dificuldades na elaboração deste ataque, uma dessas dificuldades deveu-se ao facto de não ser possível realizar o registo do "injector.exe" dentro do espaço de memória da DLL. A escrita no registo deve ser realizada com privilégios de administrador e o objectivo principal era que fosse executada no momento da injeção da DLL. Não foi possível realizar por a função ShellExecute não funcionar dentro da DLL, ou seja, não inicia o "malware.exe". De modo a contornar este obstáculo, utilizou-se a função _execl, para iniciar um outro processo (AdobeUpdater.exe) que poderá realizar a função ShellExecute com privilégios de administrador. Esta alternativa funciona mas não permite que o aviso de elevação de privilégios seja feito pelo Adobe Reader, pois o "malware.exe" não é iniciado através da DLL.

Uma dificuldade também encontrada, surge no momento da captação dos inputs do teclado e das coordenadas do rato, pois o programa captava qualquer tecla e coordenada independentemente da janela onde o utilizador se encontrasse. Isto iria originar um ficheiro (de captação) demasiado grande e com bastante informação desnecessária, então surgiu a hipótese de realizar a captação apenas na janela do IE, esta solução funciona perfeitamente no XP.

Outra dificuldade surge com a atualização na alteração na apresentação IE 9 e do IE 8 com o Windows 7 e Windows Vista. Essa alteração deve-se ao facto de as novas janelas não incluírem o nome da própria janela, ou seja o "injector.exe" procura o nome da janela que se encontra em primeiro plano e se esse nome corresponder ao "Microsoft Internet Explorer" então começa a captar os inputs. Perante a ausência do nome o "injector.exe" nunca irá encontrar a janela.



Month	Windows XP	Windows 7	Windows Vista	Mac OS X 10.6	Mac OS X 10.7	Other
June, 2011	54.04%	28.68%	10.06%	3.99%	0.02%	3.20%
July, 2011	52.80%	29.66%	9.84%	4.00%	0.33%	3.36%
August, 2011	52.46%	30.60%	9.40%	3.46%	1.03%	3.06%
September, 2011	50.50%	32.42%	9.09%	3.53%	1.41%	3.06%
October, 2011	48.03%	34.62%	8.85%	3.62%	1.83%	3.04%
November, 2011	48.89%	34.60%	8.48%	3.22%	1.86%	2.94%
December, 2011	46.52%	36.99%	8.44%	3.05%	2.02%	2.98%
January, 2012	47.19%	36.40%	8.22%	2.95%	2.18%	3.06%
February, 2012	45.39%	38.12%	8.10%	3.00%	2.69%	2.70%
March, 2012	46.86%	37.54%	7.65%	2.79%	2.59%	2.58%
April, 2012	46.08%	38.67%	7.32%	2.71%	2.71%	2.51%

Fig. 3.8 - Sistemas Operativos mais utilizados

Um dos problemas atuais é o facto de passados 11 anos desde o lançamento oficial do Windows XP, este continua a ser um dos sistemas operativos mais utilizados mundialmente. Este Sistema Operativo começa a ser substituído pelo Windows 7, no qual a implementação do UAC representa um valor acrescido na proteção contra o ataque apresentado..

3.2. Conclusão

Muito se tem feito para melhorar os sistemas operativos, procura-se atribuir aos sistemas uma garantia de confiança (ou seja se é vulnerável a certos ataques ou não) que faz parte dos requisitos de qualidade do sistema.

O objectivo da presente tese tendo como base um ataque híbrido com recurso a engenharia social, foi implementar um malware (keylogger) que permitisse verificar a evolução dos sistemas de segurança nos sistemas operativos. O programa criado consiste num malware simples, que se instala no registo do sistema operativo e regista os inputs dados no teclado pelo utilizador e as coordenadas onde o utilizador "clica".
Observou-se que o Windows XP permite sem qualquer contrariedade instalar o malware no registo sem dar algum tipo de aviso ao utilizador. O Windows Vista e 7, permitem instalar o malware no registo sem qualquer problema caso o utilizador desligue o UAC, caso contrário o SO não permitirá aplicar a DLL injection e perante isso a escrita no registo não será realizada.

Um meio possível para contornar esta contrariedade seria a aplicação maliciosa mostrar ao utilizador uma mensagem a indicar que o Windows Update não se consegue executar por conflito com o UAC e pedir para desligar temporariamente o UAC. Esta mensagem teria de incluir o aviso para desligar o computador da rede, e manter o UAC desligado não por mais de 1 minuto. Assim a aplicação já se poderia instalar no registo do sistema operativo.

Em termos de segurança constata-se que o Windows XP está muito atrás do Vista e do Windows 7, visto ser mais antigo não oferece tanta segurança, pois não está equipado com as novas proteções contra malware como os seus sucessores.

A metodologia proposta permitiu:

- Apontar um ataque que, com recurso a Engenharia Social, consegue contornar algumas das proteções técnicas convencionais; e
- Testar a eficiência das proteções atuais.

Perante estes resultados verifica-se que os sistemas operativos ainda se encontram vulneráveis assim como os utilizadores dos mesmos não estão cientes dos perigos que estão sujeitos. Alguns dos possíveis métodos que permitissem evitar estes tipos de ataques seriam:

- Compra de software original, pois garante a constante atualização do sistema operativo;
- As empresas deveriam apostar em formação sobre engenharia social aos seus colaboradores, de modo a melhorar o conhecimento sobre os cuidados que devem ter;e
- Nunca desativar o sistema UAC.

3.3. Trabalho Futuro

Como principal trabalho futuro propõem-se:

- Testar o comportamento de outros sistemas operativos tais como Mac OS X, Solaris 10, a este ataque de malware;

- Avaliar os Sistemas Operativos com outros tipos de ataque;

- Tornar o programa de ataque mais robusto e mais eficiente, de modo a conseguir contornar as proteções dos Sistemas Operativos; e

- Alterar o sistema de detecção de janela do Internet Explorer 9, de modo a que o ataque funcione a 100% no Windows 7.

4. Referências Bibliográficas

A evolução dos Microprocessadores.
Disponível: http://informatica.hsw.uol.com.br/microprocessadores1.htm
[acedido: 3 Janeiro 2012]

Amoroso, E. (1994). *Fundamentals of Computer Security Technology* .Prentice Hall Upper Saddle River, USA. Prentice Hall

Common Criteria for Information Technology Security Evaluation. (2004)
Disponível: http://www.commoncriteriaportal.org/public/expert/index.php?menu=2
[acedido: 2 Fevereiro 2012]

Correia M., Sousa P. (2010). *Proteção em Sistemas Operativos,* in FCA. *Segurança em Software,* Lisboa, FCA, pp. 35-36.

Correia M., Sousa P. (2010). *Buffer-OverFlows,* in FCA. *Segurança em Software,* Lisboa, FCA, pp. 60-73.

Correia M., Sousa P. (2010). *Corridas,* in FCA. *Segurança em Software,* Lisboa, FCA, pp. 91-94.

Damas L. (1999). *Apontadores*, in FCA. *Linguagem C*, Lisboa, FCA. Pp. 264-265

Dor N. et al. (2003). *Towards a realistic tool for statically detecting all buffer overflows in C*. In Proceedings of the ACM SIGPLAN 2003 Conference on Programming Language Design and Implementation, pp. 155–167

Feiertag .R (1980). *A technique for proving specifications are multilevel secure*. Technical Report CSL-109, Computer Science Laboratory, SRI International, Menlo Park, Canada

Fiorio M. et al. (2007).VII Simpósio Brasileiro em Segurança da Informação e de Sistemas Computacionais.

Malcolm A. (2007). *Social Engineering: A Means to Violate a Computer System*. USA

Marques J.A., Ferreira P., Ribeiro L.V, Rodrigues R., (2009), *Introdução: Evolução Histórica*, in FCA. *Sistemas Operativos*, Lisboa, FCA, pp. 8-10

Mac OS X Security Configuration (2010). USA. Apple Inc.

Maziero A. (2011). *Sistemas Operacionais: Conceitos Básicos.* Brasil

McKibben A. (2005). Introduction to Cover Channel Analysis Including Impact on Privacy.

Mean Block Rate for Socially-Engineered Malware (2010).
Disponivel:http://technology-corner.com/wp-content/uploads/2010/12/report-NSS-Labs.jpg
[acedido: 6 Julho 2011].

Microsoft Malware Protection Center (2009).
Disponível :http://blogs.technet.com/b/mmpc/archive/2009/01/13/msrt-released-today-addressing-conficker-and-banload.aspx
[acedido: 6 Julho 2011].

Microsoft Security Intelligence Report Vol. 9 (2010), USA, Microsoft

Microsoft Security Intelligence Report Vol. 10 (2010), USA, Microsoft

Microsoft Security Intelligence Report Vol. 11 (2011), USA, Microsoft

Rushby J. (1986) *The Bell and La Padula Security Model.* Computer Science Laboratory SRI International Menlo Park,USA

Smashing stack for fun and profit. Disponivel: http://insecure.org/stf/smashstack.html
[acedido: 18 Dezembro 2011].

Social Engineering Fundamentals, Part I: Hacker Tactics . (2001). Disponível: http://www.symantec.com/connect/articles/social-engineering-fundamentals-part-i-hacker-tactics
[acedido: 5 Julho 2012].

Sotirov A., Dowd M. (2008). Bypassing Browser Memory Protections: Setting Back Browser Security by 10 Years.

Support Microsoft. (2007). Disponível: http://support.microsoft.com/kb/815065
[acedido: 8 Fevereiro 2012].

Tanenbaum A. (2001).*Memory Management: Basic Memory Management*, in Prentice Hall, Horton M (Ed.). *Modern Operating Systems :Second Edition*, New Jersey, Prentice Hall, pp. 190-202

Tanenbaum A. (2001), *Security: Attacks from Inside the System*, in Prentice Hall, Horton M (Ed.). *Modern Operating Systems :Second Edition*, New Jersey, Prentice Hall, pp. 610-612

Tanenbaum A. (2001), *Processes and Threads: Interprocess Comunication, in* Prentice Hall, Horton M (Ed.). *Modern Operating Systems :Second Edition*, New Jersey, Prentice Hall, pp. 100-101

The Common Criteria. (2005). Disponível: https://buildsecurityin.us-cert.gov/bsi/articles/best-practices/requirements/239-BSI.html
[acedido: 6 Fevereiro 2012].

VIGILANTe "Social Engineering". (2001).
Disponivel: http://www.vigilante.com/inetsecurity/socialengineering.htm
[acedido: 5 Julho 2012].

Windows 7 Passes XP, Now Most Popular OS in the World (2011). Disponível: http://www.tomshardware.com/news/msft-windows-xp-windows-7-market-share-win7,13876.html
[acedido: 8 Julho 2011].

Windows Defender (2011). Disponível: http://sourcedaddy.com/windows-7/windows-defender.html
[acedido: 10 Julho 2011].

Windows 7 vs. Windows Vista UAC (2009).

Disponível: http://www.ditii.com/2009/08/05/windows-7-vs-windows-vista-uac-benchmark/

[acedido: 3 Julho 2011].