ARTHUR WONGTSCHOWSKI

SEGURANÇA EM APLICAÇÕES TRANSACIONAIS NA INTERNET: O ELO MAIS FRACO

Dissertação apresentada à Escola Politécnica da Universidade de São Paulo para obtenção do Título de Mestre em Engenharia.

São Paulo
2005

ARTHUR WONGTSCHOWSKI

SEGURANÇA EM APLICAÇÕES TRANSACIONAIS NA INTERNET: O ELO MAIS FRACO

Dissertação apresentada à Escola Politécnica da Universidade de São Paulo para obtenção do Título de Mestre em Engenharia.

Área de Concentração:

Sistemas Digitais

Orientador:

Wilson V. Ruggiero

São Paulo
2005

Este exemplar foi revisado e alterado em relação à versão original, sob responsabilidade única do autor e com a anuência de seu orientador.

São Paulo, 16 de maio de 2006.

Assinatura do autor

Assinatura do orientador

FICHA CATALOGRÁFICA

Wongtschowski, Arthur

Segurança em aplicações transacionais na Internet: o elo mais fraco/ A. Wongtschowski. – ed. rev. – São Paulo, 2005.

108 p.

Dissertação (Mestrado) — Escola Politécnica da Universidade de São Paulo. Departamento de Engenharia de Computação e Sistemas Digitais.

1. Redes de computadores (Segurança). 2. INTERNET. I. Universidade de São Paulo. Escola Politécnica. Departamento de Engenharia de Computação e Sistemas Digitais. II. t.

AGRADECIMENTOS

Agradeço ao meu orientador, Wilson Ruggiero, pelos conselhos valiosos que sempre me direcionaram para o melhor caminho. Agradeço também ao amigo e mestre Paulo Barreto, um exemplo de pessoa e de profissional a ser seguido.

RESUMO

O problema das fraudes virtuais em serviços de *Internet Banking* e comércio eletrônico vem crescendo a cada dia. Em 2005, as fraudes virtuais representaram, no Brasil, uma perda de 300 milhões de reais. Neste trabalho, apresentamos inicialmente o cenário atual das fraudes, especificando quais ataques elas utilizam, como são hoje executadas e quais vulnerabilidades exploram. Cada ataque é classificado conforme uma nova taxonomia criada. São também apresentadas possíveis defesas contra esses ataques, indicando a efetividade de cada uma. Analisamos também o ponto de vista da privacidade do usuário para cada uma dessas soluções. Como contribuição original, esse trabalho apresenta uma solução de curto prazo, que opera sobre a infra-estrutura já existente, que visa reduzir o problema das fraudes. São feitas também considerações sobre possíveis soluções de longo prazo de maior efetividade e que promovam uma defesa mais estrutural.

ABSTRACT

The problem of virtual frauds in Internet Banking and electronic commerce services is growing each day. In 2005, the virtual frauds represented a damage of 300 million reais in Brazil. In this work, we present initially the current scenario of virtual frauds, specifying what attacks are used, how they are executed and which vulnerabilities they exploit. Each attack will be classified in accordance to a new methodology. We also present defenses against these attacks, indicating the effectiveness and intrusion of each one of them. In the end, a short term and defensive solution is presented to reduce the problem of virtual frauds. We also consider long term solutions that could be used in the future to accomplish structural defenses.

SUMÁRIO

LISTA DE FIGURAS

LISTA DE TABELAS

1 INTRODUÇÃO

"Only the Paranoid Survive"

Andrew S. Grove

Segurança é um tópico que vem sendo considerado mais importante a cada dia. Bilhões de dólares são gastos todo ano em sistemas que visam garantir o perfeito funcionamento de sistemas *on-line*, como páginas de comércio eletrônico, *Internet Banking* e muitos outros serviços. Servidores mais robustos e confiáveis são comprados, sistemas de *firewall* são instalados, certificados digitais e dados criptografados são transmitidos incessantemente para garantir a completa proteção do sistema. Mas aparentemente, até agora, um ponto importante foi esquecido e deixado de lado: o cliente.

A relevância desse problema se torna simples de ser entendida se pensarmos na metáfora de uma corrente constituída de inúmeros elos. Um sistema de segurança completo é composto por vários desses elos. Exemplificando, podemos pensar em um sistema de *Internet Banking* apresentando os seguintes elos: servidor de base de dados, servidor *web*, *firewall*, sistema de *IDS* (*Intrusion Detection System*), canal de comunicação, *Internet*, e na ponta, a máquina do cliente. Imagine que nesse exemplo todo o sistema do servidor (desde a base de dados até o canal de comunicação) esteja muito bem configurado, sendo a segurança desse considerada perfeita (até utópica). Mesmo com todos esses aparatos de segurança funcionando corretamente, se, na ponta, na máquina do cliente, o elo estiver fraco, a corrente inteira poderá ser rompida. Pode

parecer exagero, mas no decorrer deste trabalho serão apresentadas diversas maneiras reais e efetivas de ataque à máquina do cliente. Algumas exploram a ignorância e desconhecimento do cliente, outras utilizam técnicas mais avançadas de invasão, mas todas têm o intuito de capturar informações críticas e secretas na própria máquina do cliente.

Como podemos então fortalecer o elo mais fraco, o do cliente? Já existem hoje alguns métodos para realizar essa tarefa, que serão expostos durante o decorrer deste trabalho. Mas, para fins de entendimento, imaginemos um caso extremo: para evitar a fraude em seu serviço de *Internet Banking*, um banco qualquer instala um certo *software* na máquina de seus clientes que passa a monitorar tudo o que por lá ocorre. Para o banco, esse pode parecer um excelente método, já que provavelmente seu número de fraudes irá reduzir consideravelmente. Mas, e quanto à privacidade do cliente? O que ele irá pensar ao verificar que agora um *software* em sua máquina pode ter acesso a todos os seus dados pessoais?

Esse é basicamente o problema que estamos tentando resolver: encontrar um meio-termo entre proteção e privacidade relativo à segurança na máquina do cliente.

1.1 Objetivos

Este trabalho tem como objetivo principal identificar, analisar e propor soluções para vulnerabilidades de segurança no ambiente cliente de um sistema Cliente-Servidor para evitar fraudes em instituições que operam sistemas transacionais na *Internet*, procurando preservar a privacidade do usuário final.

O termo “instituição transacional” será empregado neste trabalho para referenciar qualquer instituição que realize operações utilizando informações sigilosas de usuários.

1.2 Motivação

A importância de mecanismos de proteção ao cliente vem crescendo a cada dia. Podemos observar pela Figura 1.1 que o número de incidentes reportados praticamente dobra a cada ano (CERT, 2005). Sendo assim, é altamente provável que a ameaça de *malwares* (*trojans*, worms, vírus e *spywares*) continuará a crescer.

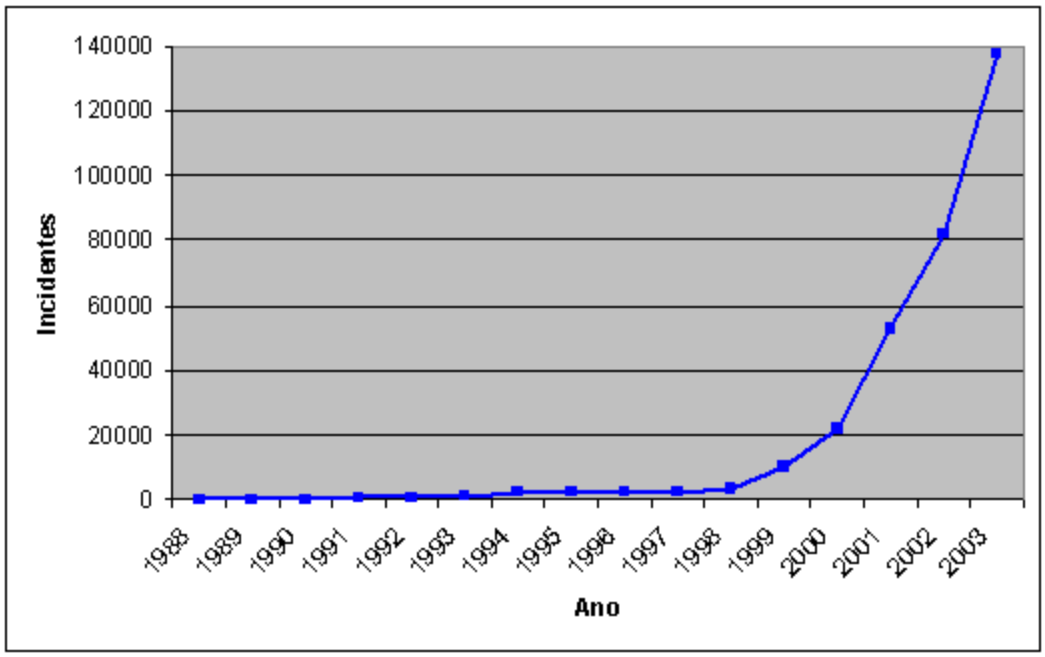

Figura 1.1: CERT: incidentes reportados.

De acordo com a Figura 1.1, podemos verificar um crescimento exponencial nas fraudes ocorridas. Sendo assim, fica claro que uma solução de curto prazo para esse problema deve ser rapidamente implementada.

Para ilustrar bem o fato, seguem abaixo algumas estatísticas extraídas de notícias publicadas em jornais e revistas do Brasil nos meses de março até julho de 2005 (IDG-NOW, 2005; FOLHA, 2005).

- As fraudes virtuais cresceram 1313% no Brasil no segundo trimestre de 2005, se comparadas com o mesmo período de 2004.

- Os golpes por *e-mail* cresceram 226% em maio de 2005 em relação ao mês anterior.
- De cada 10 *e-mails* enviados no mês de maio de 2005, mais de 3 continham vírus.
- Em março de 2005, policiais prenderam pirata brasileiro que teria desviado até US$ 37 mi.
- Os *scams* cresceram 26% por mês entre julho de 2004 e fevereiro de 2005.
- Em março de 2005, o Brasil era o quinto país em ranking de *phishing scam*.

Além disso, novas vulnerabilidades e falhas de segurança vêm sendo encontradas freqüentemente em todos os tipos de aplicativos. Também estão sendo criadas a cada dia novas maneiras de ataque aos clientes.

O conjunto de todos esses fatores tem levado pessoas de todo o mundo a ficarem preocupadas e receosas ao utilizar sistemas críticos a partir da *Internet*, como mostra a notícia abaixo:

IDG Now, 30 março de 2005: *Temor com segurança afasta europeu de e-banking*

A preocupação com ameaças de segurança como *phishing*, captura de teclado (*keystroke logging*) e outros tipos de scams está crescendo de maneira notável entre os usuários de *internet banking* na Europa ...Apenas 30% dos 22.907 usuários europeus entrevistados ...declararam confiar na segurança de suas informações financeiras pessoais quando fazem transações via *internet* ...

Como se pode verificar, os ataques específicos a clientes de *Internet Banking* vêm crescendo de forma assustadora. Com os dados apresentados acima, podemos conjecturar os seguintes pontos:

- Em nossos dias, ataques específicos ao cliente ocorrem em larga escala, pois a dificuldade técnica de sua execução é pequena (como será mostrado mais adiante), e o retorno financeiro é muito grande.

- Esses tipos de ataque continuarão a existir enquanto proverem bom retorno financeiro, mesmo que a dificuldade técnica em sua implementação aumente.

- Atualmente não é possível garantir a integridade da máquina do cliente.

- Se nada for feito, o número de fraudes tenderá a aumentar e o prejuízo das empresas e possoas tenderá a crescer.

- Existe também a propensão de os clientes abandonarem os serviços via *Internet* por preocupação com segurança.

Baseado nos pontos colocados acima, deve ficar claro que algo deve ser feito para a tentativa de preservação da segurança na máquina do cliente.

1.3 Contribuições

As contribuições apresentadas neste trabalho são as seguintes:

- Estudo de segurança de sistemas sob uma nova perspectiva: a máquina do cliente.

- Exposição de novos métodos de ataque baseados em técnicas de intrusão e seus riscos pouco conhecidos para o público em geral.

- Apresentação de métodos de proteção capazes de detectar e prevenir tais situações, minimizando o risco associado.

- Avaliação do balanceamento entre proteção e privacidade na perspectiva do cliente.

- Criação de uma nova taxonomia para os ataques direcionados ao cliente.

1.4 Estrutura do trabalho

No Capítulo 2, iremos definir uma métrica de classificação para ataques aqui explicados. O objetivo é ajudar a classificar e organizar os ataques para posteriormente facilitar a sua análise.

Já no Capítulo 3, apresentaremos as funcionalidades de segurança atuais com as quais a maioria dos sistemas se preocupa. Iremos descrever brevemente cada uma dessas funções em termos do que cada uma pretende proteger.

No Capítulo 4, serão apresentados uma variedade de métodos conhecidos para ataque direto ao cliente. Para cada um dos métodos, será citada sua complexidade técnica e efetividade de ataque.

Em seguida, no Capítulo 5, serão expostas as diferentes maneiras de proteção do cliente. Para cada uma dessas maneiras, será apresentada sua eficiência em relação aos ataques propostos, além de sua dificuldade de implementação. Além disso, será mostrado o nível de intrusão que cada um desses métodos preventivos pode vir a causar. Ao final desse Capítulo, na seção 5.4, será apresentado um esboço de solução que englobe os melhores métodos de proteção apresentados e que possa ser implantado rapidamente.

Será, então, discutido no Capítulo 6 quais as medidas futuras que podem ser tomadas para garantir, a longo prazo, a segurança dos clientes.

2 DEFINIÇÃO DO PROBLEMA

"Only two things are infinite, the universe and human stupidity, and I'm not sure about the former. "

Albert Einstein

2.1 Resumo

Neste capítulo, iremos definir exatamente o problema a ser tratado neste trabalho. Vamos definir também a abordagem utilizada no estudo do problema, além de estabelecer o foco mais preciso dos ataques e defesas estudados.

2.2 Especificação dos Ataques

O objetivo principal dos realizadores de ataques estudados neste trabalho é cometer fraudes virtuais. Isso significa basicamente tentar roubar dinheiro dos clientes de um serviço de *Internet Banking* ou comércio eletrônico. Além disso, os ataques a serem apresentados aqui são os realizados apenas na máquina do cliente, ou seja, não requerem nenhum tipo de atuação fraudulenta no servidor legítimo da instituição atacada.

2.2.1 Tipos de ataque

Existem vários tipos de ataque direcionados ao cliente. Abaixo encontra-se uma classificação geral de cada um desse tipos.

2.2.1.1 Roubo de dados

Esse ataque tem como objetivo roubar os dados críticos de autenticação do cliente. Com essas informações, o atacante pode, de forma *off-line*, reproduzir o processo de autenticação, passando-se, então, pelo cliente. Isso também é chamado de ataque de personificação. O roubo dos dados pode ocorrer de vários modos, como captura de teclado, telas falsas, *e-mails* falsos, e muitos outros, como será explicado nos capítulos seguintes.

2.2.1.2 Roubo de sessão

Algumas vezes, o roubo dos dados de autenticação do cliente não é suficiente para reproduzir sua autenticação. Por exemplo, sistemas que utilizam senhas diferentes a cada autenticação impedem que o atacante possa realizar a fraude apenas com a simples captura dos dados digitados. Nesse caso, pode ser utilizado um outro método de ataque chamado roubo de sessão. Aqui o atacante espera o cliente realizar a autenticação e após isso utiliza o número de sessão do usuário para realizar transações *on-line*. Caso cada transação também exija uma senha diferente a cada vez, o atacante pode esperar o cliente realizar uma transação para então capturar a senha daquele momento e utilizá-la na hora para realizar a fraude.

2.2.1.3 Modificação de transações

Semelhante ao ataque acima, o ataque de modificação de transações ocorre *on-line* na máquina do cliente. Nesse ataque, um aplicativo malicioso localmente instalado ficaria esperando o cliente realizar alguma transação. Nesse momento, após o cliente ter

digitado todos os dados da transação mais os seus dados de autenticação, o aplicativo malicioso poderia modificar os dados da transação e enviar esse pacote modificado. Assim, o atacante estaria apenas adulterando os dados da transação que o cliente estivesse realizando.

2.2.1.4 *Man-in-the-middle*

Os ataques de *man-in-the-middle* (ou homem-no-meio) se baseiam na idéia de interceptação dos dados durante seu tráfego entre o cliente e o servidor. Nesse tipo de ataque, o atacante consegue ler, inserir e modificar à vontade as mensagens trocadas. Esse ataque é muito importante no meio criptográfico, pois implica em grandes cuidados, principalmente em protocolos de troca de chaves. Aplicado a fraudes na *Internet*, esse ataque poderia ser utilizado para interceptar ou modificar os dados críticos trafegados entre o cliente e o servidor da instituição.

2.3 Proteção

O objetivo da proteção apresentada neste trabalho é prevenir ou ao menos reduzir a fraude. Como os ataques estudados são referentes ao lado cliente, a proteção também será apresentada na perspectiva do cliente. Nesse ponto encontramos dificuldades que não existiam no ambiente servidor, tais como:

- Como garantir a segurança em um sistema quando não se tem o controle absoluto do mesmo?

Como a máquina do cliente não faz parte do sistema sob controle da instituição, é extremamente complexo responder à pergunta acima. Um banco, por exemplo, tem total controle sobre seus servidores. Assim, é possível instalar atualizações de seguranças, aplicativos de antivírus, *firewalls* e muitas outras ferramentas de segurança para proteger esse servidor. Contudo, essa instituição não tem controle sobre a situação

das máquinas de seus clientes. Alguns clientes podem possuir antivírus, outros não. Alguns clientes têm conhecimento sobre navegação segura na *Internet*, outros não, e assim por diante.

Mesmo se a instituição instalar um *software* proprietário de segurança em cada um de seus clientes, esse *software* tem que levar em conta os diversos ambientes heterogêneos dos clientes e os diversos perfis de usuários caso participem das decisões de segurança a serem tomadas.

- Como interagir com o usuário de forma segura, ou seja, essa interação não pode ser utilizada para violar o ambiente do próprio cliente?

A instalação de um *software* de segurança pela instituição muitas vezes pode ser usada como porta de entrada por atacantes. Da mesma maneira que a instituição oficial explica aos usuários os problemas de segurança e pede então sua autorização para a instalação desse *software*, o atacante pode usar o mesmo método para tentar persuadir o usuário a instalar aplicativos maliciosos. A criação dessa interação de instalação pela instituição pode criar no usuário o hábito de permitir instalações de outros *softwares* possivelmente ilícitos.

2.3.1 Tipos de proteção

Mesmo com os problemas apresentados acima, algumas proteções ainda são possíveis e muitas vezes necessárias na máquina do cliente. Abaixo encontra-se uma classificação dos tipos gerais de proteção aos quais uma instituição pode recorrer com o objetivo de tentar reduzir as fraudes:

2.3.1.1 Prevenir contra captura de dados

O primeiro método é relativamente simples de ser entendido: prevenindo que o atacante roube as senhas do cliente, esse não poderá reproduzir o processo de autenti-

cação. Note que esse método de proteção só previne os ataques do tipo roubo de dados, sendo que os outros ataques apresentados (roubo de sessão, modificação de transações e *man-in-the-middle*) não estão cobertos por essa proteção.

2.3.1.2 Modificar os dados da autenticação

O segundo tipo de proteção requer algum tipo de senha que mude de forma imprevisível a cada operação de autenticação. Isso também é chamado de *OTP* (*One Time Password*). Nesse caso, mesmo que o atacante capture os dados críticos do cliente, esse não poderá repetir o processo de autenticação, pois no próximo processo de autenticação será requisitada outra senha. Esse método de proteção cobre somente os ataques do tipo roubo de dados, não protegendo contra os outros tipos de ataque apresentados.

2.3.1.3 Assinatura individual de transações

Nesse método de proteção, o cliente assina digitalmente os dados da transação sendo efetivada. A instituição então verifica a assinatura dos dados e só então libera a transação. Esse método de proteção é muito interessante e eficiente, porém muitos cuidados devem ser tomados durante sua implementação, como por exemplo, assinatura às cegas, validação visual dos dados assinados, local onde será realizada a assinatura, e muitos outros. Esse tópico é extremamente extenso e seria provavelmente fruto de outro trabalho completo. Esse tipo de proteção não será discutida em profundidade aqui.

É interessante notar que normalmente nenhuma das proteções citadas acima protegem totalmente o cliente contra a chamada engenharia social. O atacante sempre pode encontrar algum modo de tentar convencer o cliente a fazer o que ele deseja. Por exemplo, para um sistema que utilize uma tabela contendo diversas senhas diferentes a serem usadas na autenticação, ele poderia pedir ao cliente sua tabela completa de senhas. Neste caso, a segurança ficaria toda a cargo do cliente.

2.4 Foco do trabalho

Neste trabalho será estudado a fundo apenas um tipo de ataque e um tipo de defesa: roubo de dados e prevenção de captura, respectivamente. Os motivos para essa escolha são os seguintes:

- Atualmente, os ataques mais freqüentes são os de roubo de dados.
- O método de prevenção de captura, apesar de não ser o mais eficiente, é o mais prático e simples de ser implantado. Distribuir milhões de senhas ou dispositivos externos para cada cliente nem sempre é uma alternativa viável, principalmente pelo custo e logística envolvida. Além disso, nem todos os clientes possuem dispositivos como celulares ou *handhelds* que poderiam ser utilizados como ferramentas de autenticação.
- O cliente muitas vezes não está interessado nem disposto a complicar seu método de autenticação para melhorar sua segurança. Muitas vezes, clientes de instituições transacionais consideram que segurança é problema da instituição, não do cliente. Nem sempre é possível obrigar um cliente a carregar uma tabela de senhas ou um *token OTP*.
- Muitos clientes de instituições transacionais são leigos em relação à informática e segurança. O uso de dispositivos complexos ou métodos de autenticação complicados nem sempre é aplicável a todos os clientes. Usuários já estão acostumados com o seu esquema atual de autenticação e normalmente não vêem razão para que isso complique, atrapalhando seu uso do sistema.

Por esses motivos, a maioria dos sistemas de autenticação atual ainda se baseia na entrada de um simples par usuário/senha. Além disso, por experiência pode ser constatado que a grande maioria dos ataques atuais só se baseia na técnica de roubo de dados. Sendo assim, este trabalho será focado no estudo dos ataques de roubo de dados e nas

possíveis proteções que impeçam esse roubo. Isso obviamente não descarta a importância ou eficiência dos outros tipos de ataques e proteções anteriormente citados, mas isso seria assunto para outro trabalho.

Iremos também focar esse trabalho no sistema operacional *Windows*, pois sua utilização hoje é maioria absoluta, tornando-se assim o principal alvo de ataques.

2.5 Trabalhos relacionados

Alguns outros trabalhos já foram escritos sobre classificação e taxonomia de ataques. Em (LANDWEHR ALAN R. BULL; CHOI, 1994), os autores descrevem uma taxonomia completa para falhas de segurança de computadores. Em (HOWARD, 1997), é descrita uma taxonomia de ataques a redes de computadores. Em (WARKENTIN XIN LUO, 2005), é descrito um arcabolço para classificação de *spywares*.

Existem muitas publicações sobre o tópico de fraudes em sistemas transacionais na *Internet*. Recentemente, várias publicações tem abordado esse assunto de formas diferentes. Alguns exemplos: em (DHAMIJA, 2005), os autores propõem uma solução para dificultar os ataques de *phishing*. Em (GIBSON, 2005), o autor apresenta o problema atual dos *spywares* e o porque do seu grande crescimento. Em (LEE, 2005), os autores expõe os fatores mais relevantes na escolha de um sistema de proteção contra *spywares*.

Vários outros trabalhos citados no texto explicam o problema de segurança envolvido nesse ambiente transacional e tentam identificar algumas soluções mais estruturais.

Apesar disso, a área de publicações ainda é vaga quanto à parte mais técnica de descrição dos ataques e proteções aqui apresentadas. Muitas referências mais abrangentes sobre codificação existem espalhadas pela *Internet* ou em documentações da própria *Microsoft*, mas em nenhum ponto foi encontrada a apresentação do perigo

na utilização maliciosa de certas técnicas de programação, nem na utilização dessas mesmas técnicas com o intuito de prevenção. O conhecimento e a utilização dessas técnicas existem mais pela necessidade de atacantes e instituições. O conhecimento empírico de alguns profissionais, obtido através de experiência nessa área, é normalmente a única fonte de informação acessível.

2.6 Taxonomia dos ataques

Como já foi dito, o objetivo final dos realizadores dos ataques aqui apresentados é cometer uma fraude virtual. No caso mais específico dos ataques de roubo de dados, seu objetivo é capturar de alguma maneira os dados críticos do cliente (como suas senhas) para poder num futuro replicar o processo de autenticação.

Os ataques de captura normalmente devem seguir os seguintes passos para serem bem-sucedidos:

1. Instalação - O atacante deve de alguma maneira instalar um *software* malicioso ou modificar de alguma forma a máquina do cliente.

2. Captura - Com acesso à maquina do cliente, o atacante deve agora capturar os dados críticos da vítima.

3. Envio dos dados roubados - O atacante deve então enviar os dados capturados para conseguir ter acesso aos mesmos.

4. Fraude - O atacante utiliza os dados capturados para realizar a fraude.

Nem todos os ataques aqui descritos seguem todos os passos citados cima. Alguns não precisam de instalação local de *software*, outros não precisam enviar os dados para o atacante. Mas, de forma genérica, a maioria dos ataques realiza os passos acima. Neste trabalho não serão descritos o terceiro e o quarto passos citados acima, por sua relativa simplicidade.

A seguir apresentaremos uma proposta de taxonomia para classificação desses ataques ao cliente. Cada ataque será classificado quanto às suas características principais e mecanismos de ação.

2.6.1 Tipo do ataque

Inicialmente, definiremos qual o tipo do ataque. Conforme explicado acima, um atacante deve normalmente seguir alguns passos para conseguir realizar a fraude. Alguns ataques estão focados em apenas um dos passos citados. Outros podem ser ataques completos, que sozinhos já conseguem realizar totalmente seu objetivo.

Os ataques deste trabalho terão seus tipos classificados da seguinte maneira:

- Ataque completo: ataque que, sozinho, já realiza por completo seu objetivo. Não requer o auxílio de nenhum outro ataque.
- Ataque parcial: ataque que realiza apenas um ou alguns dos passos necessários para completar totalmente o objetivo do atacante. Sozinho, não realiza a fraude, mas pode ser utilizado em conjunto com outros ataques.

Um ataque completo de captura pode ser constituído de vários ataques parciais que somados resultam na fraude.

2.6.2 Objetivo imediato do ataque

O objetivo final dos ataques aqui descritos é sempre o de propiciar aos realizadores de ataques a chance de realizar uma fraude. Contudo, cada ataque parcial possui seu objetivo individual. O objetivo imediato dos ataques parciais aqui apresentados serão classificados da seguinte maneira:

- Instalação: ataque cujo objetivo é instalar algo na máquina do cliente.
- Captura de dados: ataque cujo objetivo é obter os dados críticos do cliente.

2.6.3 Local do ataque

Nessa etapa, especificaremos o local onde o ataque ocorre, ou seja, o módulo utilizado para a realização do ataque apresentado. Esses locais podem ser os seguintes:

- Aplicativos gerais: aplicativos de terceiros, não fundamentais ao sistema, utilizados pelo usuário (exemplos: aplicativo de gerenciamento financeiro, aplicativo de edição de texto).
- Aplicativos de sistema: aplicativos do próprio sistema, fundamentais para o dia-a-dia (exemplos: *e-mail*, navegador *web*).
- Sistema operacional: código do próprio sistema operacional, obrigatório para o correto funcionamento do sistema (exemplos: serviços do sistema, *DLLs* do sistema);
- Núcleo do sistema: parte central do sistema operacional também conhecida como *kernel* (exemplo: *drivers* de periféricos).

Quanto mais baixo o nível do ataque, normalmente mais complicada é sua implementação.

2.6.4 Vulnerabilidade explorada pelo ataque

Aqui classificaremos qual vulnerabilidade específica é utilizada pelo atacante para realizar o ataque, ou seja, qual falha está sendo explorada pelo atacante para que o ataque seja bem sucedido. As vulnerabilidades podem ser classificadas conforme a lista abaixo:

- Falha de implementação: o sistema atacado possui um *bug*, ou seja, a implementação do sistema possui um defeito que pode ser explorado para a execução

do ataque. Exemplo: vulnerabilidades do sistema operacional exploradas pelos *worms Sasser* e *Blaster* (apresentados mais a frente).

- Defeito de especificação: o sistema foi especificado de maneira a deixar uma brecha de segurança que pode ser explorada por atacantes. Isso não quer necessariamente dizer que o sistema foi mal implementado ou que possui um *bug*. Isso simplesmente quer dizer que o sistema foi mal arquitetado. Exemplo: funções desnecessárias disponibilizadas pelo sistema operacional que auxiliam na captura de dados.

- Inexperiência do usuário: fator humano, vítima da engenharia social. Um usuário inexperiente tem maior chance de ser vítima de um ataque que um usuário mais experiente. Exemplo: *e-mails* falsos com *links* contendo programas maliciosos.

2.6.5 Complexidade técnica do ataque

Como normalmente o ataque apresentado envolve codificação (programação), ou pelo menos algum conhecimento técnico sobre a área, podemos classificar os ataques pela sua dificuldade de implementação. Esse ítem é de extrema importância, pois está ligado diretamente à escolha do ataque a ser utilizado pelo atacante. Inicialmente, atacantes procurarão ataques de menor complexidade técnica, pois sua execução é mais simples. Conforme as proteções e dificuldades para os ataques forem aumentando, atacantes começarão a migrar para ataques mais complexos.

Neste trabalho iremos classificar os ataques conforme sua dificuldade técnica, utilizando a seguinte escala:

- Ataque simples: não requer conhecimento técnico profundo nem grandes trabalhos de programação. Esses ataques muitas vezes podem ser realizados sem

programação nenhuma, pois o atacante procura na *Internet* programas já prontos e apenas os modifica superficialmente.

- Ataque mediano: requer razoável conhecimento técnico sobre o assunto, incluindo os mecanismos internos do sistema operacional. Não requer necessariamente grandes quantidades de programação e muitas vezes pode ser realizado apenas com linguagens de alto nível.

- Ataque complexo: requer conhecimento técnico profundo e extenso trabalho de programação. É necessário conhecer detalhes do sistema operacional, incluindo funcionalidades raramente utilizadas ou até não documentadas. Pode inclusive requisitar programação em linguagem de máquina.

De uma maneira geral, ataques mais técnicos que exijam conhecimento profundo sobre o funcionamento do sistema operacional serão mais complexos do que ataques que possam ser realizados apenas com linguagens visuais de programação. Ataques que exijam modificação no núcleo do sistema também serão mais complicados. Códigos disponíveis na *Internet* e facilmente adaptáveis serão mais simples.

2.6.6 Transparência do ataque

Para o cliente, alguns ataques ocorrem de forma mais transparente que outros. Ataques que utilizam programas escondidos que capturam as informações digitadas de forma furtiva são mais transparentes que aqueles que mostram janelas na tela do usuário pedindo todos os seus dados. Usuários mais leigos podem ser vítimas dos dois ataques citados, mas provavelmente usuários mais experientes perceberiam a presença do segundo ataque. Nesse sentido, apresentamos a classificação da transparência dos ataques conforme abaixo:

- Baixa transparência: o ataque modifica bastante a forma normal esperada da aplicação original.

- Média transparência: o ataque modifica pouco a forma normal esperada da aplicação original.
- Alta transparência: o ataque modifica muito pouco ou nada da forma normal esperada da aplicação original.

Essa classificação indica basicamente quão simples (ou quão complicado) é a detecção pelo usuário de que este está sendo atacado num dado instante. Quanto mais transparente um ataque, mais complexa é sua detecção pelo usuário.

Esse item é utilizado apenas para ataques de captura de dados, não sendo relevante para os ataques com o objetivo de instalação.

2.7 Ataques mais relevantes

O estudo de todos os tipos de ataques a clientes é extremamente extenso e foge do escopo deste trabalho. Aqui, decidimos estudar a fundo alguns dos mais relevantes ataques, a saber:

- *Trojans*
- Links falsos
- Exploração de vulnerabilidades
- *Keyloggers*
- Telas falsas
- Ataque de redirecionamento
- Ataque à máquina virtual *Java*

O motivo principal da escolha desses ataques é que, com exceção do último, esses são hoje os ataques mais freqüentes realizados a sistemas de *Internet Banking* (CERT, 2005). Além disso, esses ataques são também muito efetivos hoje na captura dos dados do cliente. Também é relevante o fato de esses ataques serem tecnicamente mais simples, o que aumenta bastante sua atratividade para os atacantes.

O ataque à máquina virtual *Java* foi também selecionado, pois utiliza uma nova técnica de captura que será explicada mais à frente. Essa nova técnica é extremamente poderosa e pode potencialmente ser utilizada para investir contra outros tipos de máquinas virtuais. Além disso, o ambiente *Java* é muito utilizado hoje por instituições durante o processo de autenticação de seus usuários, e é exatamente nesse ponto que os atacantes irão atacar para capturar os dados do cliente.

3 CENÁRIO CLIENTE/SERVIDOR

3.1 Preocupações atuais com segurança: Servidor

Neste capítulo apresentaremos resumidamente algumas preocupações atuais com segurança no ambiente cliente-servidor. Citaremos algumas proteções usadas principalmente no lado servidor e especificaremos em que ponto se encontra a proteção dos clientes.

3.1.1 SSL

As preocupações com a confidencialidade das informações trafegadas pela *Internet* vem de velha data. O *SSL* (*Secure Sockets Layer*) é um protocolo desenvolvido pela *Netscape* para transmissão de documentos pela *Internet*. Esse protocolo utiliza em conjunto criptografia de chave simétrica e criptografia de chave assimétrica para garantir a integridade e confidencialidade, fim a fim, dos dados transportados.

O *SSL* é muito utilizado por páginas que obtêm informações confidenciais de clientes, como número de cartões de crédito ou senhas de *Internet Banking*. Atualmente a utilização de *SSL* é quase obrigatória em todos os serviços que exijam confidencialidade nas informações transmitidas.

Como já foi dito, o *SSL* garante a comunicação fim a fim. Apesar disso, na versão 2.0 do *SSL* (a mais amplamente utilizada atualmente) não existe identificação do cliente perante o servidor, ou seja, apesar de o cliente conseguir validar a autenticidade do

servidor, o contrário não é verdade. Além disso, o *SSL* só protege os dados a partir do momento em que são enviados pelo navegador através da rede, sendo que entre a digitação e o momento do envio, os dados permanecem em claro.

3.1.2 *IDS*

Para evitar ataques ou invasões de sistemas, foram criados os chamados *IDS* (*Intrusion Detection System*). O trabalho desse sistema é inspecionar toda a atividade da rede (tanto os dados de entrada quanto os dados de saída) e identificar padrões suspeitos que possam indicar a ocorrência de um ataque. Os sistemas de *IDS* podem ser classificados como:

- Detecção por ataques conhecidos ou por anomalia: No primeiro caso, o *IDS* possui uma base de dados de ataques conhecidos, e portanto consegue procurar dentro do tráfego de rede ataques semelhantes aos cadastrados, representando um sistema semelhante a um antivírus. No segundo caso (detecção por anomalia), o sistema passa um tempo em modo aprendizado, em que deve aprender qual o comportamento normal daquela rede. Depois de um certo tempo, o sistema consegue detectar anomalias na rede a partir da base de dados criada.

- Baseado em rede ou em *host*: *IDS* baseados em rede monitoram cada pacote de comunicação individualmente. *IDS* baseados em *host* monitoram a atividade em cada máquina individualmente.

- Passivo ou reativo: *IDS* passivos, ao detectar um possível ataque, gravam a informação e ativam algum tipo de alerta. Em um sistema reativo, o sistema de *IDS* exerce uma ação no caso de uma anomalia, como, por exemplo, derrubar algum usuário ou passar a bloquear todos os pacotes vindos do endereço suspeito.

Sistemas de detecção de intrusão são muito importantes para o ambiente servidor, pois evitam ataques que poderiam provavelmente derrubar um sistema inteiro. Mesmo

assim, esses sistemas só são interessantes para o ambiente servidor, sendo inviável a sua instalação em clientes individuais.

3.1.3 *Firewall*

O objetivo de uma *firewall* é prevenir acesso não autorizado a uma rede privada. *Firewalls* podem ser implementadas tanto em *software* quanto em *hardware*, ou mesmo em uma combinação dos dois. O objetivo mais comum de uma *firewall* é impedir que usuários não autorizados da *Internet* acessem redes privadas (*intranets*). Uma *firewall* pode, por exemplo, filtrar os pacotes de uma rede, examinando pacote por pacote entrando ou saindo, podendo então aceitar ou rejeitar o tráfego baseado em regras definidas pelo usuário. Os sistemas de *firewall* são considerados a primeira linha de defesa na proteção de redes privadas.

Apesar de relacionados com segurança de redes, um *IDS* difere de uma *firewall* no sentido que a *firewall* limita o acesso de usuários para prevenir intrusões, além de não proteger contra ataques internos da rede. Um *IDS*, por sua vez, verifica uma possível intrusão depois que ela já ocorreu, e sinaliza algum alarme. Um *IDS* também pode detectar ataques oriundos da rede interna.

3.1.4 Atualizações de segurança

Novas vulnerabilidades de segurança são encontradas em todos os tipos de produtos a cada dia. Só em abril de 2005, foram lançados vinte atualizações de segurança pela *Microsoft* para corrigir falhas encontradas em seus produtos. Apesar de o número de falhas de segurança em produtos tender a diminuir, elas sempre existirão e portanto serão exploradas por atacantes.

A instalação de atualizações críticas de segurança deveria ocorrer com freqüência, mas para a maioria dos clientes isso não ocorre. Seja por mero esquecimento, seja pelo número elevado de atualizações de segurança ou qualquer outro motivo, usuários

comuns normalmente não aplicam essas correções em suas máquinas, ficando assim vulneráveis a ataques já conhecidos e corrigidos.

Dois exemplos impressionantes podem ser dados para ilustrar essa situação: o *worm Sasser* e o *worm Blaster*. O *Blaster* explora a vulnerabilidade de *DCOM* do *RPC* no *Windows 2000*. A *Microsoft* estima que pelo menos 8 milhões de computadores com *Windows* foram infectados pelo *Blaster*. Já o *Sasser* explora uma vulnerabilidade mais recente no *Windows*, espalhando-se de máquina em máquina sem a intervenção do usuário. A *Microsoft* reportou que mais de 1.5 milhões de clientes baixaram o programa de remoção do *Sasser* nos primeiros dois dias após sua disponibilização. Até hoje, mesmo após meses da liberação do *patch* de correção pela *Microsoft*, se um usuário conectar um micro sem *firewall* e sem as atualizações críticas na *Internet*, é quase certo que após alguns minutos sua máquina será infectada por algum desses *worms* citados.

3.2 Preocupações atuais com segurança: Cliente

Atualmente, a preocupação de segurança com o cliente ainda é muito pequena. Apesar de a segurança dos servidores da *Internet* normalmente já estar bem evoluída e consolidada, o lado cliente permanece esquecido. De qualquer maneira, já existem algumas poucas investidas na tentativa de proteção ao cliente, principalmente em páginas de *Internet Banking*.

4 MÉTODOS DE ATAQUE AO CLIENTE

4.1 Resumo

Neste capítulo serão apresentados os ataques mais relevantes realizados contra a máquina do cliente. Cada ataque será classificado conforme a metodologia estabelecida em capítulos anteriores. Também será apresentada uma explicação técnica da implementação de cada um dos ataques. As propostas de defesa para esses ataques serão apresentadas no Capítulo 5.

4.2 Advertência ética

As informações sobre os métodos de ataque estão aqui apresentadas somente para fins didáticos. Não é o intuito deste trabalho ensinar como se pode atacar um sistema transacional. O objetivo aqui é apresentar de forma resumida os ataques mais conhecidos, tornando mais simples a implementação de suas respectivas defesas. É sempre necessário entender como podemos ser atacados antes de tentarmos nos defender.

4.3 Modo de instalação

Nesta seção são apresentados alguns exemplos de ataques que têm como objetivo a instalação de um programa malicioso na máquina do cliente sem que este perceba.

4.3.1 *Trojans*

Na tabela 4.1, apresentamos a classificação do ataque *trojans* conforme a taxonomia criada. Nesse caso, o item "transparência" não foi apresentado, pois só se aplica para métodos diretos de ataque (e não para métodos de instalação).

Trojans	
Tipo	parcial
Objetivo	instalação
Local	aplicativo de sistema (*e-mail*)
Vulnerabilidade	inexperiência
Complexidade	simples

Tabela 4.1: Classificação do ataque *trojans*.

Inicialmente, vamos definir o que é um *trojan* (ou cavalo de Tróia). *Trojans* são programas malignos disfarçados de programas benignos. O *trojan* se utiliza da ignorância, curiosidade ou falta de informação do cliente para conseguir se instalar em sua máquina (AARONSON, 2005). A partir do momento em que o *trojan* consegue se instalar no sistema, ele passa a ter total controle sobre a máquina e pode então monitorar todas as atividades do usuário.[1]

Atualmente, o modo mais comum de instalação de um *trojan* é a partir de *e-mails*. O método é o seguinte: o atacante monta um *e-mail* que incita a curiosidade ou atenção do cliente. Esse *e-mail* é fabricado de forma a induzir o cliente a, inadvertidamente, instalar o *trojan* em sua máquina.

Na Figura 4.1, o atacante tenta se utilizar da curiosidade do usuário para que este clique no *link* esperando fotos interessantes, quando na verdade o *link* leva a um executável que instalará um *trojan* na máquina desse usuário.

[1]Para simplificação da terminologia, o nome *trojan* será também utilizado neste trabalho para referenciar genericamente qualquer *software* malicioso instalado localmente na máquina do cliente

Figura 4.1: Exemplo de e-mail contendo *trojan*.

Outros exemplos de *e-mails* contendo *trojans* podem ser vistos no Apêndice, nas Figuras A.1, A.2, A.3 e A.4. Todos esses *e-mails* lá apresentados são reais e foram recebidos sem nenhuma autorização prévia. Pode não parecer, mas todos eles são falsos e levam à baixa de um *trojan*.

Depois de instalado, o *trojan* utilizará uma das técnicas descritas na seção 4.4 para capturar as informações do usuário. Após isso, os dados capturados serão enviados a algum servidor, provavelmente por *FTP* (*File Transfer Protocol*), *HTTP* (*Hypertext Transfer Protocol*) ou *SMTP* (*Simple Mail Transfer Protocol*), onde o atacante poderá coletá-los.

Além do uso do *e-mail* como veículo de transmissão do *trojan*, outros métodos

ainda podem ser usados. Um exemplo seria anexar o *trojan* a algum programa lícito, como jogos ou aplicativos de transmissão de arquivos (*pear to pear*).

4.3.2 Vulnerabilidades

Na tabela 4.2, apresentamos a classificação do ataque vulnerabilidades conforme a taxonomia criada. Nesse caso, o item "transparência" não foi apresentado, pois só se aplica para métodos diretos de ataque (e não para métodos de instalação).

Vulnerabilidades	
Tipo	parcial
Objetivo	instalação
Local	sistema operacional
Vulnerabilidade	erros de implementação
Complexidade	média

Tabela 4.2: Classificação do ataque vulnerabilidades.

Como já foi citado em seções anteriores (3.1.4), a exploração de vulnerabilidades do sistema operacional e aplicativos é um modo comum para instalação de *software* de forma ilícita na máquina do cliente. Como explicado para os *worms Sasser* e *Blaster*, um aplicativo malicioso pode explorar essas vulnerabilidades para instalar um remotamente *trojan* na máquina da vítima.

Atualmente, esse método ainda não é muito utilizado para a instalação de *trojans* específicos para ataques, provavelmente pela complexidade técnica em sua execução.

4.4 Modo de captura

Nesta seção serão apresentados algum métodos mais comuns para executar a captura das informações críticas de clientes. As possíveis defesas contra esses ataques serão apresentadas no Capítulo 5.

4.4.1 *Links* falsos

Na tabela 4.3, apresentamos a classificação do ataque *Links* falsos conforme a taxonomia criada.

Links falsos	
Tipo	completo
Objetivo	fraude
Local	aplicativo de sistema (*e-mail*)
Vulnerabilidade	vulnerabilidade de especificação e inexperiência do usuário
Complexidade	simples
Transparência	baixa

Tabela 4.3: Classificação do ataque *links* falsos.

Outra maneira muito comum de ataque ao cliente é o envio de *e-mails* em nome de alguma instituição transacional, contendo um *link* para um *site* falso muito semelhante ao da instituição real. O *e-mail* normalmente contém alguma informação sobre débitos a sanar, ou mesmo sobre promoções mirabolantes da instituição. Ao clicar no *link* dentro do *e-mail*, o usuário é levado a um *site* falso onde serão pedidos seus dados críticos, como agência, conta e senhas. Ao final, o *site* falso pode simplesmente apresentar um erro ou mesmo redirecionar o usuário ao *site* verdadeiro, tornando assim o ataque mais transparente.

4.4.2 *Keyloggers*

Na tabela 4.4, apresentamos a classificação do ataque *Keyloggers* conforme a taxonomia criada.

Keyloggers são programas que capturam os dados provindos do teclado, *mouse* e tela. A maioria dos *keyloggers* capturam tudo o que é digitado pelo cliente durante seu acesso a algum *site* de *Internet Banking*. Além disso, com o número crescente de bancos utilizando teclados virtuais para realizar a autenticação de seus usuários, os *keyloggers* também são projetados para capturar pedaços de tela ao redor do cursor do

Keyloggers		
Tipo		parcial
Objetivo		captura
Local		sistema operacional
Vulnerabilidade		vulnerabilidade de especificação
Complexidade	*Hooks*	média
	Funções estáticas	média
	Captura de campos	média
	Drivers	alta
	DLL injection	alta
Transparência		alta

Tabela 4.4: Classificação do ataque *keyloggers*.

mouse durante os *cliques*, sendo assim possível a recuperação da senha do cliente.

A seguir são expostos alguns possíveis métodos de captura de teclas e *cliques*, focando em sistema operacional *Windows*.

4.4.2.1 *Hooks*

Hooks (ganchos) do sistema operacional podem ser utilizados tanto para a captura de *mouse* quanto para a captura de teclado. Em termos técnicos, os *hooks* são pontos dentro do sistema de tratamento de mensagens do *Windows* (MSDN, 2005) em que um aplicativo pode instalar uma sub-rotina para monitorar o tráfego de mensagens no sistema ou processar certos tipos de mensagens antes de estas chegarem à sua janela destino. Em termos mais simples: a comunicação de eventos entre processos do sistema operacional *Windows* (por exemplo, um clique em um botão, a digitação de uma tecla, o fechamento de uma janela) funciona por meio de troca de mensagens. Quando um usuário clica no botão *OK* de uma aplicação, o sistema operacional envia uma mensagem para aquela aplicação avisando o evento do clique. A aplicação recebe então essa mensagem e a trata conforme necessário. *Hooks* são pontos intermediários que um aplicativo pode registrar junto ao sistema operacional para receber algum tipo de mensagem antes que ela seja entregue à janela correta.

O mesmo tipo de *hook* pode ser registrado por várias aplicações diferentes. Nesse caso, o sistema operacional passa a mensagem para o primeiro *hook* na fila, que, por sua vez, deve repassá-lo para o segundo *hook*, e assim por diante, conforme a Figura 4.2, onde se pode verificar também que o *hook* fica alojado entre o sistema operacional e a aplicação final que receberá a tecla.

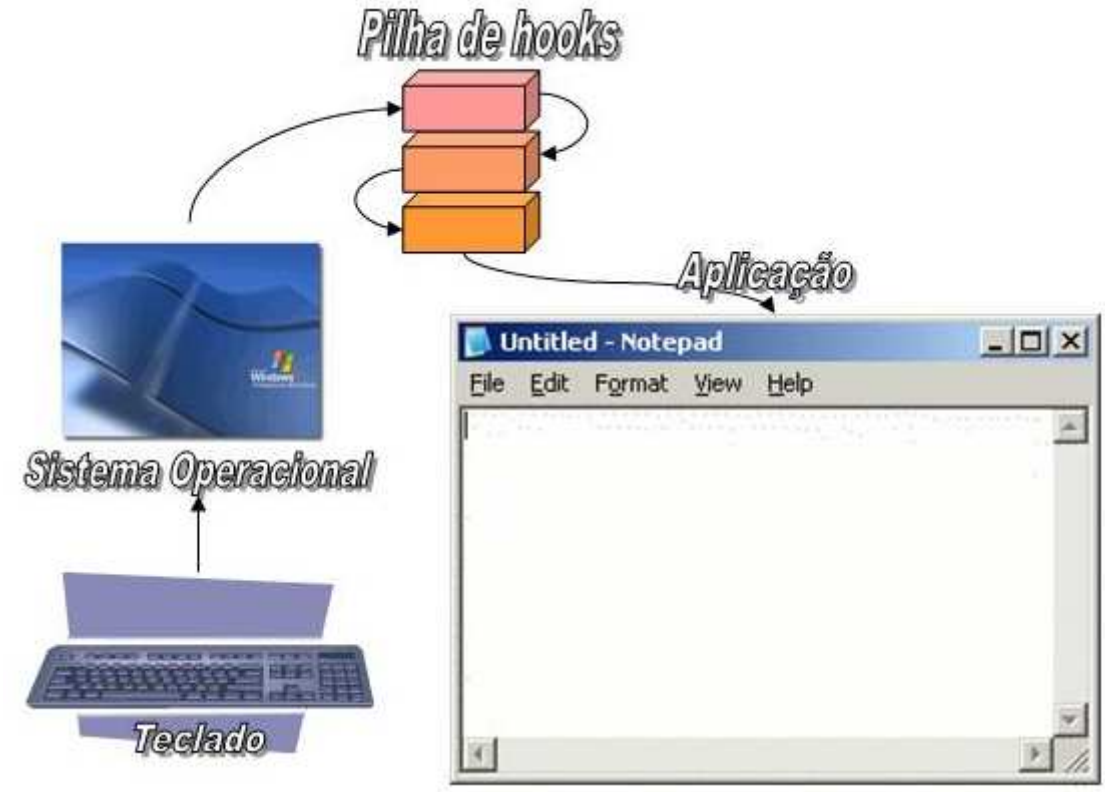

Figura 4.2: Hooks: caminho percorrido por uma tecla.

Dentro de uma rotina de *hook*, o sistema operacional repassa alguns parâmetros importantes, como as coordenadas do clique do *mouse*. Nesse instante, fica simples realizar a captura desse clique. Por exemplo, caso o usuário esteja clicando em um teclado virtual para a entrada de sua senha no banco, a rotina de *hook* poderia utilizar as coordenadas do clique para tirar uma foto do pedaço da tela onde ocorreu o clique, roubando assim a senha do usuário, conforme Figura 4.3 (teclado virtual extraído de (SMARTLINK, 2005)).

É interessante notar que nesse momento a janela final que efetivamente recebeu

(a) Teclado virtual para entrada de dados (b) Tela capturada ao clicar na letra *i*

Figura 4.3: Exemplo de captura de clique de *mouse* utilizando *hooks*

o clique do *mouse* ainda não está a par desse clique, pois a rotina de *hook* recebe a mensagem antes de essa ser processada. Portanto, métodos de proteção amplamente utilizados, como, por exemplo, sumir com os números do teclado virtual no momento do clique, não protegem contra esse método de captura.

Existem vários tipos de *hook* que podem ser utilizados para captura de teclas digitadas ou para captura de *cliques* de *mouse*, todos baseados no processo descrito anteriormente: monitoração de mensagens enviadas e recebidas por janelas.

A disponibilização de uma ferramenta como os *hooks* pode ser considerada uma grave falha na especificação do próprio sistema operacional. Apesar de o mecanismo de *hooks* ser bastante utilizado por aplicações lícitas, sua concepção leva a crer que o quesito segurança foi deixado de lado durante seu desenvolvimento. Os *hooks* tornam extremamente simples a monitoração completa de todas as mensagens trocadas na máquina do usuário, e não existe nenhuma maneira de desabilitar esse comportamento. A simples existência de um mecanismo com essa funcionalidade já pode colocar em risco a segurança do sistema.

4.4.2.2 Funções estáticas do teclado

Existem no *Windows* funções estáticas que podem ser usadas de forma assíncrona para o obtenção do estado atual do teclado. Nessas funções deve-se passar um parâmetro que especifique o código virtual da tecla que está sendo consultada. Analisando o retorno dessas funções, é possível saber se a tecla consultada está ou não pressionada no momento da chamada. Um *trojan* pode utilizar esse método para capturar todas

as teclas digitadas pelo usuário, utilizando essas funções estáticas dentro de um laço, passando por todas as teclas que interessarem. A Figura 4.4 apresenta de forma gráfica o funcionamento assíncrono dessas funções.

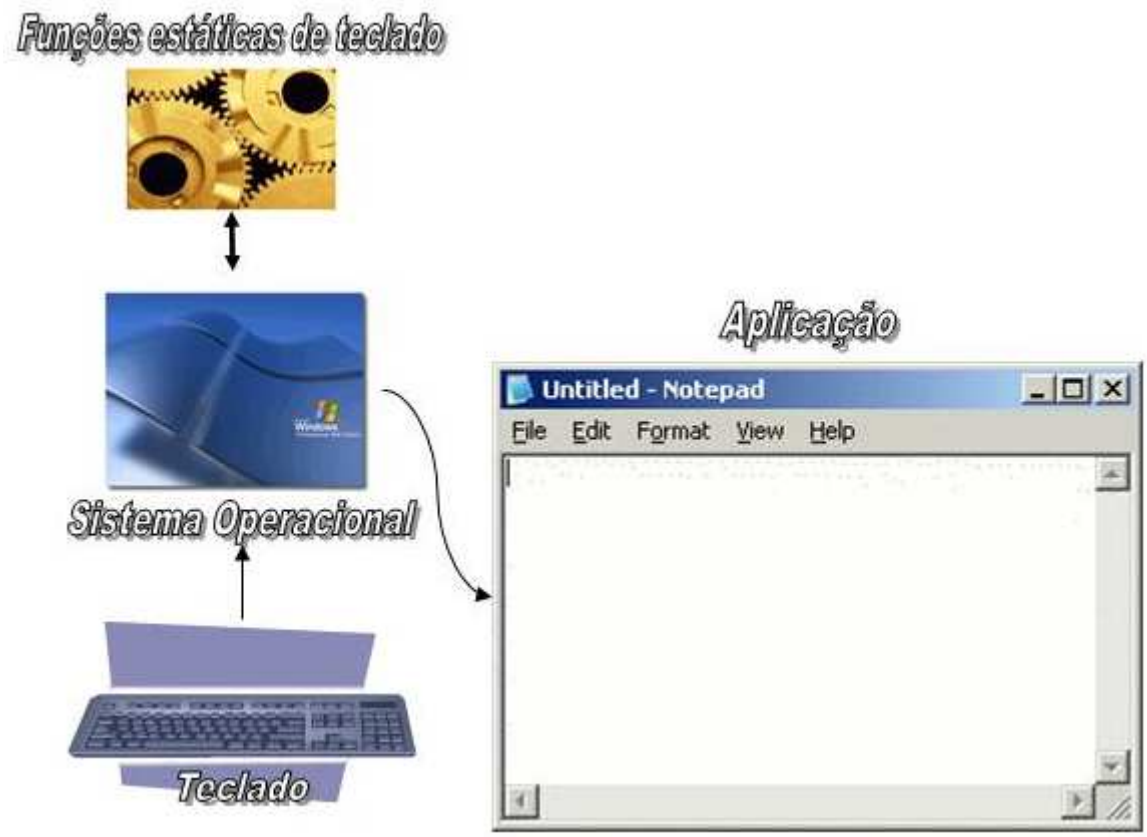

Figura 4.4: Funções estáticas: caminho percorrido por uma tecla.

A grande vantagem da utilização de funções estáticas em vez das funções de *hook* é a maneira assíncrona em que essas funções trabalham. Nas funções de *hook*, o processo que recebia o clique do *mouse* ou a tecla digitada tinha que esperar a rotina de *hook* executar para então poder receber a tecla. Isso podia afetar a performance em sistemas lentos ou carregados. Além disso, era necessário escrever uma rotina de *hook* externa para que essa pudesse funcionar de forma global para todos os processos. No caso das funções estáticas, um simples executável com 15 linhas de código já é suficiente para realizar a captura. Por outro lado, pela sua natureza assíncrona de captura, não se pode garantir que o *trojan* que utiliza essa técnica não perca algumas teclas caso o usuário

as digite muito rapidamente ou mesmo capture algumas teclas repetidas caso o usuário as pressione por muito tempo.

4.4.2.3 Captura de campos de texto

Apesar de não ser tecnicamente um *keylogger*, a captura de campos de texto é muito prática e difundida na *Internet*. Nesse caso, a idéia não é capturar o que o usuário digita no momento em que está pressionando o teclado, mas sim capturar os dados digitados quando já estiverem contidos num campo de texto do aplicativo destino. O exemplo da Figura 4.5 mostra uma tela simples de um sistema qualquer que requisita que o usuário digite sua senha:

Figura 4.5: Tela para entrada de senha

Durante a digitação, os caracteres não aparecem na tela, evidentemente para tentar proteger a senha. Assim, caso um *trojan* tire uma foto da tela nesse momento, ele não conseguirá descobrir nada a não ser talvez o tamanho da senha do usuário.

Para esconder a senha, o sistema operacional *Windows* utiliza um truque simples que nada mais é do que trocar o caractere digitado por um caractere fixo, chamado de *PASSWORDCHAR*. Por padrão, o *PASSWORDCHAR* utilizado é o asterisco. Caso um outro aplicativo mande uma mensagem para essa janela pedindo o texto digitado num campo do tipo senha, o sistema operacional sabe que não deve repassar nada e só retorna uma mensagem em branco.

Para passar essa proteção, os aplicativos de captura de campos de texto enviam uma mensagem para a janela-alvo pedindo para que essa troque seu *PASSWORDCHAR* para zero (o número 0). Ao receber essa mensagem, o comportamento-padrão de qual-

quer janela é verificar que o parâmetro passado é zero e então desabilitar a proteção da senha. Nesse momento, se o usuário digitar mais alguma coisa no campo, tudo aparecerá em claro. Para evitar esse problema, os aplicativos de captura mandam logo em seguida uma mensagem para o campo da senha pedindo o seu valor. Como agora o *PASSWORDCHAR* é zero, o sistema operacional não acha mais que aquele campo é restrito, portanto ele repassa a senha completa para quem a pedir. Para terminar, o aplicativo de captura manda uma mensagem final para a janela-alvo pedindo para que ela retorne seu *PASSWORDCHAR* para o asterisco. Assim, o usuário não percebe que o aplicativo de captura conseguiu roubar a senha.

Existe um *software* muito conhecido e gratuito na *Internet*, chamado *SnadBoy* (SNADBOY, 2000), que realiza o processo descrito para a captura de senhas. A Figura 4.6 mostra a tela do *SnadBoy* capturando a senha de um aplicativo de teste.

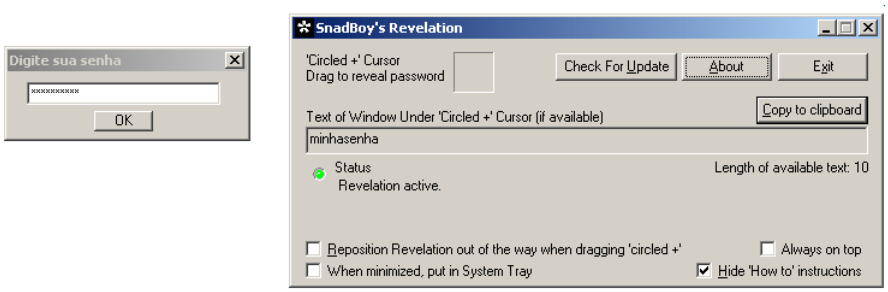

Figura 4.6: Captura da senha do usuário

4.4.2.4 *Drivers*

Drivers fazem parte da camada mais baixa que liga o *hardware* da máquina com o sistema operacional. Numa instalação-padrão do *Windows*, *drivers* de teclado e *mouse* já vêm instalados e configurados para que possamos utilizar de forma transparente esses periféricos. Mas, internamente, existe uma complexa cadeia de eventos que ocorrem entre o bater de uma tecla e a chegada dessa tecla no aplicativo de destino.

O desenvolvimento de *drivers* para Windows segue um modelo chamado *WDM*

(*Windows Driver Model*). Dentro desse modelo, um *driver* pode ser construído conforme três tipos distintos: *Bus drivers*, *Function Drivers* e *Filter Drivers* (ver Figura 4.7)

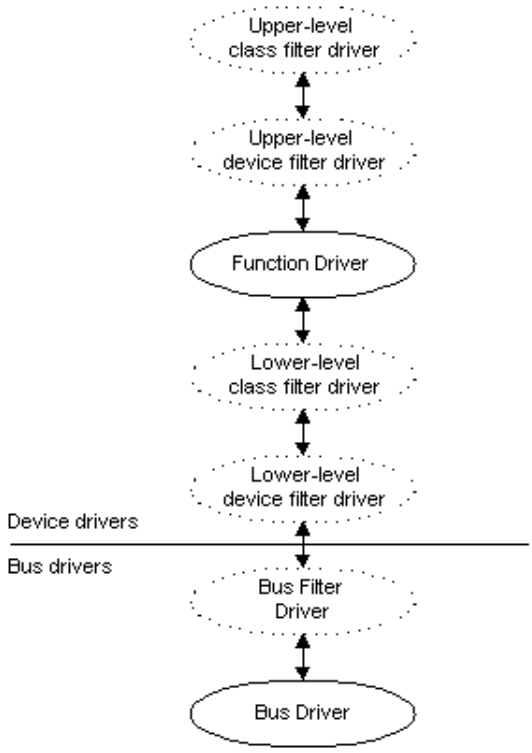

Figura 4.7: Relação entre os vários tipos de *drivers*.

- *Bus Drivers*: controlam uma via de entrada e saída e provêm funcionalidade para cada placa conectada a essa via de forma transparente ao dispositivo conectado. Exemplos: *PCI*, *ISA*, *AGP*

- *Function Drivers*: controlam um dispositivo específico. Exemplos: *driver* de teclado ou de *mouse*.

- *Filter Drivers*: filtram as requisições pendentes de entrada e saída para um dispositivo, uma classe de dispositivos ou mesmo uma via.

Como indicado na Figura 4.7, o caminho normal que uma requisição de entrada

e saída percorre desde sua geração até sua recepção pelo sistema operacional é o seguinte:

Bus Drivers → *Bus Filter Drivers* → *Lower-level device filter drivers* → *Lower-level class filter drivers* → *Function Driver* → *Upper-level device filter drivers* → *Upper-level class filter driver*.

Todos os *Filter drivers* são opcionais nesse processo. A diferença entre um *class filter drivers* e um *device filter driver* é que o primeiro filtra requisições de uma classe de dispositivos (por exemplo, todos os discos rígidos da máquina) enquanto o segundo filtra apenas requisições específicas para um dispositivo (por exemplo, um teclado *Microsoft* proprietário).

Um teclado normal possui internamente um micro controlador 8042 que fica constantemente varrendo a matriz de linhas e colunas do teclado para verificar se alguma tecla está pressionada (HYDE, 2003). Quando o usuário aperta uma tecla, o micro controlador detecta e envia o chamado *scan code* dessa tecla para o computador. Cada *scan code* representa uma tecla diferente do teclado. Também é gerado um *scan code* diferente quando a tecla é liberada.

Quando os *scan codes* chegam ao computador, outro micro controlador realiza a conversão desses *scan codes* para a tecla respectiva, coloca o resultado na porta de entrada e saída número 60h e então gera uma interrupção, deixando a cargo do *ISR* (*Interrupt Service Request*) tratar a tecla a partir desse momento.

Olhando num nível mais alto, o que chega para o *driver* instalado na máquina já é o *scan code* corretamente convertido. A partir desse momento, é função do *driver* tratar esse *scan code* e convertê-lo corretamente para a tecla sendo digitada. Esse *driver* deve, por exemplo, verificar o tempo que uma tecla fica pressionada e gerar a seqüência corretamente; verificar o pressionamento de teclas como *shift*, *caps-lock*, *num-lock* e tratar o conjunto dessas teclas com o resto do teclado.

Voltando agora à perspectiva de captura de teclado utilizando *drivers*, um atacante teria as seguintes alternativas:

- Reescrever o *device driver*: seria teoricamente possível reescrever o *device driver* de um teclado e substituí-lo na máquina atacada. Mas como a complexidade técnica de um *driver* de teclado é razoavelmente grande e como existe uma grande variedade de layouts e tipos diferentes de teclados, essa opção acaba sendo pouco prática.

- Escrever um *filter driver*: aqui temos várias alternativas de *filter drivers*: *lower-level*, *upper-level*, *class filter* ou *device filter*. Em qualquer um dos casos, o *filter driver* espião poderia armazenar todas as teclas que passam por seu processamento, e depois apenas repassa-las para o filtro de cima na cadeia, permitindo o funcionamento correto do teclado da vítima.

O *driver* mais simples de ser escrito seria um *upper-filter class driver*. Ou seja, um *driver* que é apenas um filtro (não há necessidade de tratar a mensagem), que ficaria alocado acima do *driver* de teclado da máquina (assim as teclas passando pelo filtro já estariam corretamente tratadas) e que seria genérico para qualquer tipo de teclado (daí o nome *class driver*).

A implementação de um *driver* desse tipo não é de maneira nenhuma trivial. É necessário trabalhar no nível do *kernel* (núcleo do sistema operacional) e é preciso verificar todos os possíveis casos de erro, pois qualquer problema em um *driver* pode causar o travamento total da máquina. A Figura 4.8 mostra a localização de um possível *filter driver* espião dentro do caminho percorrido por uma tecla.

4.4.2.5 *DLL Injection*

A técnica de *DLL injection* pode ser utilizada para diversos fins, benignos ou malignos. Vamos inicialmente tentar descrever tecnicamente o queremos dizer por *DLL*

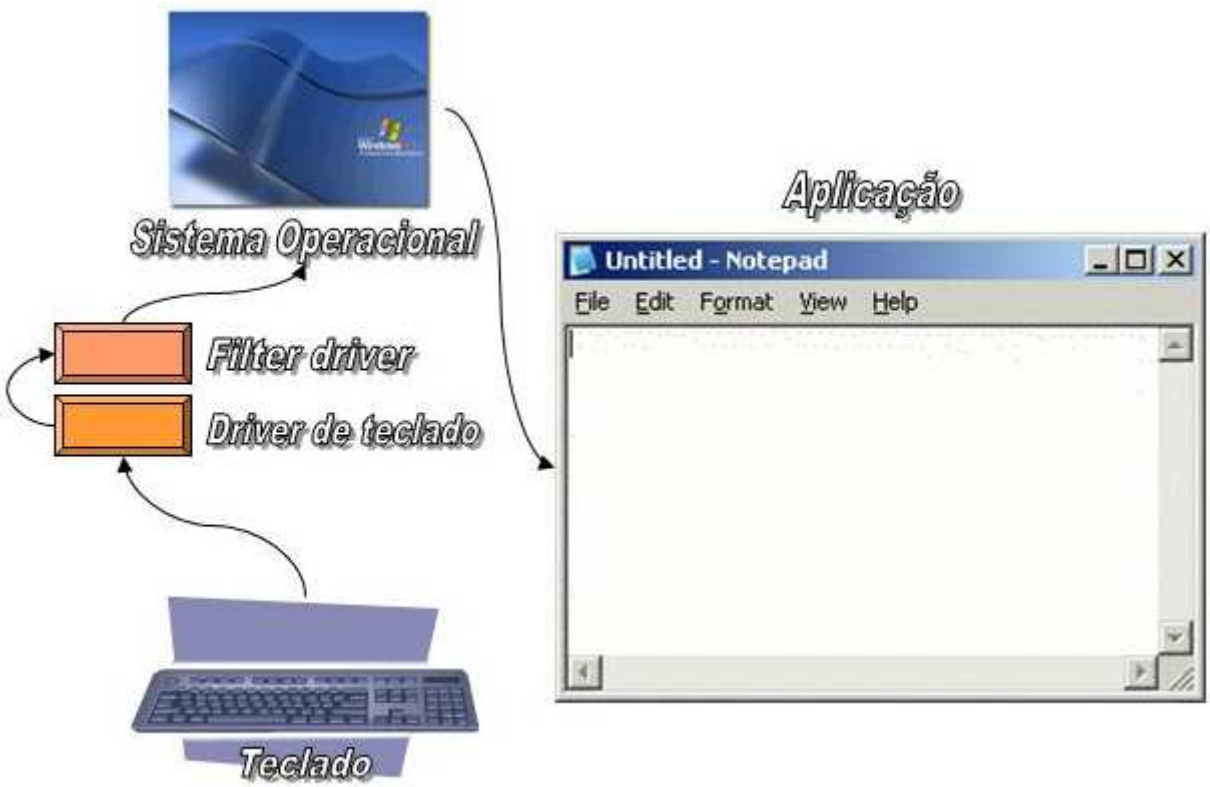

Figura 4.8: *Drivers*: caminho percorrido por uma tecla.

Injection.

A idéia dessa técnica, como o nome já diz, é conseguir carregar uma *DLL* (*Dynamic Link Library*) dentro de um processo-alvo. O objetivo é conseguir obter acesso à memória do processo vítima, algo que um programa externo não conseguiria. Caso, por exemplo, um *trojan* tentasse ler algum valor residente na memória de um outro processo, o sistema operacional detectaria uma violação de memória e finalizaria o *trojan*. Por essa razão se faz necessário carregar a *DLL* atacante dentro da memória do próprio processo vítima. Se isso for realizado com sucesso, essa *DLL* terá acesso a toda a memória daquele processo.

A partir do ponto em que o *trojan* tenha conseguido injetar uma *DLL* dentro de um processo-alvo, ele pode agora ler toda a memória do processo, pode acessar variáveis internas, sobrescrever funções e muito mais. O *trojan* pode, por exemplo, sobrescrever

a rotina de tratamento de mensagens do aplicativo. Ele poderia assim monitorar todos os eventos de teclado ou *mouse* que chegam para a janela vítima. Para manter o funcionamento da janela intacto, o *trojan* pode ainda repassar as mensagens para a rotina de tratamento original após o seu processamento.

4.4.3 Telas falsas

Na tabela 4.5, apresentamos a classificação do ataque Telas falsas conforme a taxonomia criada.

Telas falsas	
Tipo	parcial
Objetivo	captura
Local	aplicativo de sistema
Vulnerabilidade	inexperiência do usuário
Complexidade	simples
Transparência	baixa

Tabela 4.5: Classificação do ataque telas falsas.

Um ataque muito comum que não envolve a captura de teclado é o de telas falsas. O ataque basicamente tenta enganar o usuário mostrando uma tela muito semelhante à do *site* oficial, normalmente no momento em que o cliente está acessando o serviço correto. Essas telas normalmente se sobrepõem à tela do navegador de forma a parecer que fazem parte do *site* oficial.

Essas telas falsas são altamente disseminadas principalmente pelo fato de sua implementação ser bem mais simples que a implementação de um *keylogger*. Apesar de não ser totalmente transparente, esse tipo de ataque acaba sendo muito efetivo ou pela pura ignorância do cliente, ou pela refinada implementação da tela falsa.

Nas Figuras 4.10, 4.11 e 4.12, apresentamos uma simulação de um ataque desse tipo. A Figura 4.9 apresenta o *site* oficial de um banco de teste, sem nenhuma modificação. Na Figura 4.10, apresentamos um exemplo de tela falsa bem rudimentar

Figura 4.9: *Site* oficial do banco teste.

Figura 4.10: Tela falsa simples atacando um banco teste.

atacando diretamente o *site* do banco. Na Figura 4.11, apresentamos uma tela falsa um pouco mais refinada que tenta colocar sua tela exatamente em cima da tela original, dando a impressão de que nada mudou. Na Figura 4.12, apresentamos então uma tela falsa mais sofisticada que se coloca de forma estratégica exatamente em cima do campo de senha. Notamos que, mesmo sabendo da existência da tela falsa, é difícil

descobrir exatamente em qual dos campos de senha a tela se colocou.

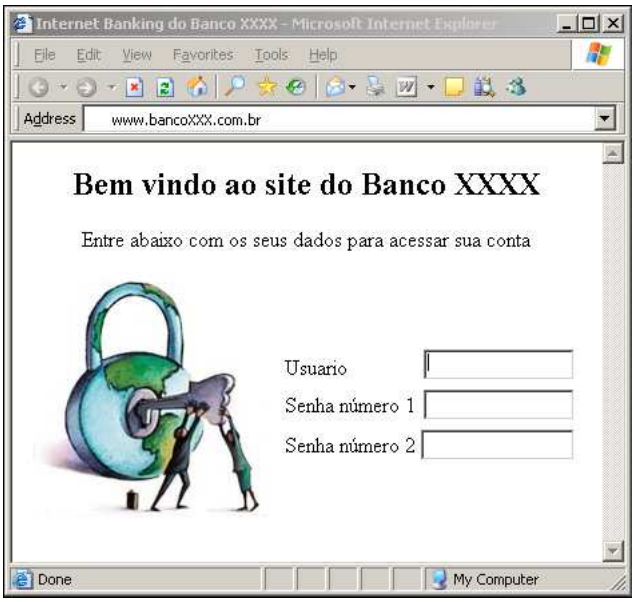

Figura 4.11: Tela falsa de média complexidade atacando o banco teste.

Figura 4.12: Tela falsa de maior complexidade atacando o banco teste.

Os ataques de telas falsas estão hoje extremamente disseminados. A complexidade desse tipo de tela vem crescendo a cada dia. Até mesmo um usuário experiente operando a máquina pode ter dúvidas quanto à legitimidade do *site*. Sua detecção é complexa, pois esse tipo de tela se assemelha muito a aplicativos comuns que a maioria

dos usuários possui em suas máquinas.

4.4.4 Ataque de redirecionamento

Na tabela 4.6, apresentamos a classificação do ataque de redirecionamento conforme a taxonomia criada.

Ataque de redirecionamento		
	Ataque ao servidor *DNS*	**Ataque local**
Tipo	completo	parcial
Objetivo	captura	captura
Local	sistema operacional (do servidor de *DNS*)	sistema operacional
Vulnerabilidade	vulnerabilidade de implementação	vulnerabilidade de especificação
Complexidade	complexa	simples
Transparência	média	média

Tabela 4.6: Classificação do ataque de redirecionamento.

Toda a comunicação na Internet se baseia no protocolo de rede *IP* (*Internet Protocol*). Esse protocolo estabelece um endereço único para cada máquina ligada à rede. Sendo assim, quando um computador deseja se comunicar com outro, basta saber o *IP* do destino e a mensagem será entregue corretamente.

Mas, como endereços *IP* não são muito amigáveis para os olhos humanos, foi inventada a técnica de resolução de nomes. Nessa técnica, o usuário final apresenta como endereço não mais o *IP* do destino, mas sim o seu nome. Um servidor centralizado fica responsável por manter uma tabela indicando qual *IP* pertence a qual nome. Assim, para o usuário, a tradução entre nome e *IP* ocorre de forma transparente.

Na *Internet*, esse servidor de resolução de nomes (chamados *DNS* - *Domain Name System*) não é uma única máquina. Existem milhares de servidores *DNS* espalhados pela rede, cada um responsável por resolver o nome de alguns servidores específicos. Por exemplo, um provedor de acesso à *Internet* possui um servidor *DNS* próprio que será utilizado apenas por seus clientes.

Como também é impossível um único servidor de *DNS* armazenar todos os nomes e *IPs* existentes na *Internet*, esses servidores funcionam dentro de uma hierarquia vertical. Caso um servidor não conheça um nome específico, ele pode perguntar ao servidor *DNS* pai se este o conhece, e assim por diante, até que alguém consiga resolver o nome. Por exemplo, quando o usuário digita em seu navegador a *URL* (*Uniform Resource Locator*) www.bancoXXXX.com.br, inicialmente essa requisição será encaminhada para o *DNS* do provedor de acesso desse usuário. Caso esse servidor não consiga resolver esse nome, ele poderá encaminhar a pergunta para o servidor *DNS* do Banco XXXX.

Aplicando esses conhecimentos, podemos enumerar três possíveis ataques:

1. Ataque direto ao servidor de *DNS*: caso um atacante consiga acesso privilegiado a um servidor *DNS* (provavelmente através da exploração de vulnerabilidades desse servidor), esse pode modificar as tabelas de resolução de nome do próprio servidor para modificar o *IP* resolvido para um nome específico. Ou seja, o atacante pode modificar o *IP* para qual o nome *www.bancoXXXX.com.br* será resolvido. Esse *IP* poderia conter um *site* falso, idêntico ao *site* oficial. Assim o atacante conseguiria capturar os dados do cliente de forma quase transparente. Note que esse ataque afetaria todos os clientes que utilizam o servidor *DNS* atacado. Além disso, não é necessária a instalação de nenhum tipo de *trojan* no cliente atacado. Ou seja, mesmo clientes com sua segurança local intacta podem estar suscetíveis a esse tipo de ataque.

2. Ataque ao sistema de resolução de nomes da própria máquina do usuário: existem, dentro do sistema operacional *Windows*, alguns módulos e arquivos de configuração que podem armazenar localmente tabelas de resolução de nomes. Durante uma requisição de resolução de nome, o sistema operacional verifica dentro desses arquivos de configuração se esse nome já não se encontra resolvido. Isso é feito antes da requisição ser encaminhada para o servidor *DNS*. Um desses ar-

quivos de configuração se chama *hosts*, e nada mais é do que um arquivo texto contendo *IPs* e respectivos nomes. Segue abaixo um exemplo desse arquivo:

```
127.0.0.1       localhost
192.168.1.2     www.acme.com.br
```

Dado que o atacante conseguiu instalar um *trojan* localmente na máquina da vítima, esse poderia inserir uma linha nesse arquivo, impedindo assim que o sistema consulte o servidor *DNS* para a obtenção do *IP* correto.

```
127.0.0.1       localhost
192.168.1.2     www.acme.com.br
1.2.3.4         www.bancoXXXX.com.br
```

Com o arquivo *hosts* configurado conforme o código acima, no momento em que o cliente digitar *www.bancoXXXX.com.br*, o sistema automaticamente o redirecionará para o *IP 1.2.3.4* sem nem mesmo consultar o servidor *DNS*. O lugar referente a esse endereço *IP* poderia conter um *site* falso, semelhante ao *site* oficial, onde o atacante capturaria as informações criticas do cliente. Novamente, esse ataque pode ser realizado de forma quase transparente para o usuário.

3. Troca de *URL* no navegador do cliente: esse método de ataque é semelhante ao anterior, no sentido de que o *trojan* deve primeiro conseguir se instalar na máquina do cliente antes de continuar com o ataque. Mas, nesse ataque, não há modificação nenhuma nos arquivos de configuração do sistema operacional. O ataque ocorre da seguinte forma:

   - O *trojan* fica escondido, monitorando todas as *URLs* que o navegador do usuário acessa.
   - Quando o usuário acessar a *URL* a ser atacada (por exemplo, *www.bancoXXXX.com.br*), o *trojan* redireciona o navegador para uma página localmente instalada na própria máquina do cliente. Essa página, que

provavelmente foi trazida junto com o *trojan*, é bem semelhante à do banco sendo atacado.

- Depois que o usuário digita seus dados críticos dentro da página falsa, o *trojan* redireciona o cliente novamente para o *site* original, tornando assim o ataque quase transparente.

4.4.5 Ataque à máquina virtual *Java*

Na tabela 4.7, apresentamos a classificação do ataque à máquina virtual *Java* conforme a taxonomia criada.

Ataque à máquina virtual *Java*	
Tipo	parcial
Objetivo	captura
Local	aplicativo gerais
Vulnerabilidade	vulnerabilidade de especificação
Complexidade	média
Transparência	alta

Tabela 4.7: Classificação do ataque à máquina virtual *Java*.

A máquina virtual *Java* atua como uma camada entre o sistema operacional e os aplicativos *Java*. A utilização da máquina virtual *Java* é especialmente útil por dois motivos:

1. O mesmo código *Java* pode ser executado em diferentes plataformas, desde que exista uma máquina virtual *Java* compatível naquele ambiente.

2. A máquina virtual *Java* tem o poder de controlar completamente o código que ela executa.

Em termos de segurança, o segundo item citado é o mais importante. Essa funcionalidade de isolamento e restrição de aplicativos em execução é chamada de *sandbox*. A máquina virtual *Java* sempre executa aplicativos não confiáveis dentro da *sandbox*.

O programa em si pode tentar executar o que bem desejar, mas todas as suas operações estarão limitadas pela *sandbox*. Aplicativos *Java* em execução dentro da *sandbox* não podem realizar as seguintes ações:

- Ler ou escrever arquivos do disco
- Estabelecer conexões através de uma rede
- Criar novos processos
- Carregar dinamicamente novas bibliotecas
- Chamar funções nativas do sistema operacional

Evitando que códigos não confiáveis executem essas operações, o modelo de segurança do *Java* protege o usuário local contra programas maliciosos. Dentro do modelo de segurança do *Java*, existe uma classe especialmente importante chamada *Security Manager*. O papel do *Security Manager* é controlar quem pode realizar certas operações. Quando um aplicativo *Java* tenta utilizar uma função mais perigosa (uma função que potencialmente poderia levar a um ataque), como por exemplo, uma função de acesso a disco, a própria *API* (*Application Programming Interface*) *Java* em execução pergunta ao *Security Manager* se esse processo pode ou não realizar essa ação. Se o *Security Manager* verificar que essa operação não é permitida, ele emite um erro e a execução dessa função é abortada. Toda a lógica de bloqueio realizada pelo *sandbox* está implementada dentro da classe *Security Manager*. Essa classe está embutida dentro das próprias bibliotecas de *run-time* do *Java*.

O ataque descrito a seguir serve como exemplo para demonstrar como é possível modificar arbitrariamente o comportamento da máquina virtual *Java*. Os passos necessários para realizar o ataque são os seguintes:

1. Abrir a biblioteca de *run-time* do *Java* (que nada mais é do que um arquivo compactado) que contém a classe *Security Manager*

2. Descompilar a classe *Security Manager*

3. Modificar a classe *Security Manager*, retirando algumas validações

4. Recompilar a classe *Security Manager*

5. Compactar novamente a biblioteca de *run-time* do *Java*

6. Substituir a biblioteca de *run-time* do *Java* pela adulterada

Durante o passo 3, podemos retirar alguma verificação de proteção do *Security Manager*, quebrando o *sandbox* e possibilitando assim que aplicativos *Java* possam se comportar de maneiras anteriormente não permitidas.

O ataque descrito acima pode ser aplicado diretamente para captura de dados críticos de clientes. Muitas páginas de *Internet Banking* e comércio eletrônico utilizam os chamados *applets de autenticação*. Esses *applets* têm como objetivo melhorar a segurança do cliente durante seu processo de autenticação. Exemplos desses *applets* são: *applets* para entrada de senhas; teclados virtuais; *applets* de criptografia; *applets* que implementam autenticação utilizando certificação digital; *applets* que implementam autenticação por biometria; entre muitos outros possíveis.

Utilizando o ataque descrito, um *trojan* localmente instalado na máquina do cliente pode criar uma maneira relativamente simples e multiplataforma para capturar dados de usuários. Para atacar um *applet* de entrada de senha, o *trojan* poderia substituir a classe que recebe a senha do cliente. A nova classe poderia armazenar a senha do cliente em disco e depois repassar a senha para a classe original, mantendo assim o ataque totalmente transparente. Para *applets* de criptografia, o *trojan* poderia trocar a própria classe que implementa a cifra criptográfica, armazenado a senha em aberto antes de essa ser encriptada. Para agilizar esse processo, o *trojan* poderia vir já com as classes a serem trocadas pré-compiladas. Isso tornaria o ataque mais rápido e prático.

O ponto forte desse ataque é sua forma totalmente transparente de execução. Mesmo usuários experientes utilizando a máquina infectada terão dificuldade em notar alguma diferença no processo de autenticação.

4.5 Considerações finais

Os ataques aqui apresentados não precisam necessariamente ser utilizados de forma isolada. É sempre possível, e talvez até mais efetivo, juntar dois ou mais ataques para torná-los mais fortes. Por exemplo, um *keylogger* pode utilizar a técnica de *hooks* junto com a técnica de *DLL Injection*, tornando assim mais efetiva sua captura dos dados críticos.

5 MÉTODOS DE PROTEÇÃO AO CLIENTE

5.1 Resumo

Neste capítulo são apresentadas as possíveis defesas que devem proteger os clientes dos ataques explicados anteriormente. Para cada técnica de proteção descrita, será explicado exatamente qual ataque está sendo inibido e quais as vantagens e desvantagens do uso dessa proteção.

5.2 Considerações iniciais

Não é um trabalho simples tentar defender o cliente dos ataques citados nas seções precedentes. Como a máquina do cliente está fora do controle da instituição que opera o sistema transacional, é difícil poder contar com qualquer tipo de premissa de segurança nesse ambiente. A máquina do cliente é um campo de batalha desconhecido. Alguns clientes podem cuidar bem de sua segurança, instalando sistemas de antivírus e *firewalls*, enquanto outros podem simplesmente usar o computador exatamente como ele veio de fábrica.

A criação de um *software* a ser executado em um ambiente tão inóspito deve ser feita com extremo cuidado.

Nos próximos capítulos, o nome *módulo de segurança* será utilizado para referenciar um *software* instalado localmente na máquina do cliente e que tem como objetivo bloquear a ação dos atacantes. Esse módulo de segurança pode ser também um con-

junto de *softwares*, incluindo a configuração adequada de cada um deles.

A instalação local de um *software* na máquina do cliente traz consigo sérias questões de segurança e privacidade. Como um usuário pode estar seguro de que esse módulo de segurança faz somente o que ele se propõe a fazer? Como garantir que esse módulo seja originário de uma instituição idônea? Como garantir que a privacidade do cliente não estará sendo invadida com esse *software* que pode possivelmente estar realizando qualquer tipo de monitoração em sua máquina?

A instalação local desse módulo de segurança pode acostumar o cliente mais leigo a aceitar qualquer tipo de instalação de *software* em sua máquina. A propaganda da instalação desse módulo de segurança pode até ser usada por *trojans* como fonte de material para realizar um *spam* mais direcionado, ou seja, um *trojan* poderia vir disfarçado tentando se passar pelo módulo de segurança.

Todas essas questões são mais políticas do que técnicas. As respostas para essas perguntas fazem parte de um escopo mais global de aquisição de conhecimentos e conscientização dos clientes quanto à navegação segura e segurança na *Internet*. Neste capítulo nos limitaremos a tratar dos aspectos mais técnicos referentes aos ataques e proteções possíveis. Os tópicos referentes às perguntas acima serão tratados em maior detalhe no Capítulo 6.

5.2.1 Batalhas em uma guerra injusta

A batalha que ocorre na máquina do cliente entre atacantes e os programas de proteção é extremamente injusta. A proteção sempre foi mais complexa que o ataque, mas nesse caso específico a diferença é ainda maior. Abaixo seguem alguns cuidados que um *software* de proteção deve ter, mas com os quais um *software* malicioso não precisa necessariamente se preocupar.

- Compatibilidade: como o ambiente de máquinas de clientes é normalmente

muito heterogêneo, um *software* de proteção deve estar preparado para funcionar perfeitamente em todos esses ambientes. Em nenhum momento o *software* pode apresentar um erro, ou ainda impossibilitar o cliente de exercer alguma função em sua máquina. Os programas maliciosos não têm necessariamente essa preocupação. Um atacante pode confeccionar um *trojan* que seja efetivo somente em um sistema operacional especifico. Clientes que possuam outro sistema operacional não serão afetados. O *trojan* pode até mostrar um erro na tela ao ser instalado nesses outros sistemas operacionais, pois o atacante não precisa se preocupar com possíveis reclamações de clientes. Além disso, se o *trojan* não foi muito bem testado e causar um erro grave em algumas situações, novamente o atacante não precisa se preocupar com isso. Desde que o *trojan* tenha bons resultados em um número razoável de máquinas, o ataque já será bem-sucedido.

- Consumo de recursos: muitas vezes, programas maliciosos baixam e enviam grande quantidade de dados pela *Internet*, consomem grande quantidade de *CPU* (*Central Processing Unit*), memória e até espaço em disco. Um *trojan* pode baixar muitos *megabytes* da *Internet* sem encontrar problemas. O cliente poderá até notar que sua conexão está mais lenta, mas mesmo assim ele não terá para quem reclamar. Muitas vezes, para realizar a captura de teclado do cliente, um *keylogger* pode consumir quase toda a *CPU* da máquina para garantir que não irá perder nenhuma tecla. Um programa de proteção não pode fazer nada do gênero. Um programa de proteção não pode esgotar os recursos da máquina do cliente, não pode deixar a máquina sobrecarregada e não pode deixar a conexão com a *Internet* lenta.

- Identificação do oponente: existem hoje milhares de variantes de programas maliciosos na *Internet*. O crescimento de *trojans* está cada vez mais se aproximando do crescimento de vírus: a cada dia surge um novo. Para um programa de proteção, é difícil identificar na máquina do cliente quais processos são ma-

liciosos e quais não são. Como um antivírus, esse sistema de proteção terá que ser constantemente atualizado para tentar manter um nível razoável de proteção. Por outro lado, um *trojan* que ataca uma instituição especifica sabe exatamente qual o programa de proteção que aquela instituição utiliza. Muitas vezes, os atacantes podem até ser clientes dessa instituição. Dessa maneira, o atacante sabe qual programa estará executando na máquina do cliente que tentará bloquear seu acesso. Sendo assim, o atacante tentará provavelmente desinstalar o *software* de segurança antes de realizar seu ataque.

- Distribuição: realizar a distribuição em larga escala de um *software* de proteção não é tarefa fácil. Clientes muitas vezes podem estranhar que a instituição está querendo instalar um *software* em sua máquina. Clientes irão reclamar, ligar para centrais de atendimento, mandar *e-mails*, e muito mais. E a cada atualização do *software* (que com certeza será necessária e freqüente), o cliente novamente será atrapalhado e advertido. Por outro lado, por mais incrível que pareça, a distribuição de programas maliciosos é extremamente simples. Caso o atacante queira distribuir um novo tipo de *trojan*, ele só tem que criar um *e-mail* chamativo, colocar um *link* nesse *e-mail* para o *trojan*, e enviá-lo para o máximo de pessoas possível. Obviamente, muitas das pessoas que receberem esse *e-mail* irão ignorá-lo, mas os poucos que abrirem e executarem o *trojans* já estarão infectados. Se posteriormente o atacante resolver mudar de alguma maneira o seu mecanismo de ataque, basta o envio de outro *e-mail*. O atacante não almeja que todos os clientes instalem seu *software*. O atacante não se preocupa com os clientes que irão reclamar do *e-mail* recebido. Para o atacante, basta que alguns instalem o *trojan* que o ataque já será bem-sucedido.

5.2.2 Prevenindo a desinstalação

Como dito anteriormente, caso alguma instituição operadora de um sistema transacional deseje disponibilizar um *software* de proteção para seus clientes, um dos primeiros problemas a serem tratados será a prevenção da desinstalação desse módulo. Da mesma maneira que muitos vírus tentam desabilitar sistemas conhecidos de antivírus, *trojans* tentarão desinstalar esse novo módulo de proteção. Normalmente, é até mais fácil desinstalar um sistema de proteção do que tentar ultrapassar suas proteções.

Inicialmente vamos definir o que é uma desinstalação: basicamente, um *trojan* será bem-sucedido em desinstalar um sistema de proteção se conseguir de alguma maneira fazer que este pare de executar seus mecanismos de proteção, seja imediatamente ou mesmo numa próxima sessão (como por exemplo, na próxima reinicialização da máquina). De maneira prática, para realizar a desinstalação de um módulo de segurança um *trojan* poderá tentar:

1. Apagar os arquivos pertencentes ao módulo
2. Apagar as chamadas de inicialização do módulo (como chaves no *registry*)
3. Desconfigurar o módulo

Prevenir a desinstalação nada mais é do que impedir que um *trojan* realize os passos acima. Mas essa tarefa é mais árdua do que parece.

Do ponto de vista técnico, existem várias maneiras que um programa pode usar para tentar prevenir sua desinstalação. É inviável listar extensivamente todos esses métodos, portanto foram escolhidos apenas os mais simples para apresentar somente a idéia de prevenção de desinstalação. Esses métodos são:

- Manter os arquivos importantes do módulo sempre em uso, fazendo com que o próprio sistema operacional impeça a sua remoção.

- Utilizar a técnica de injeção de *DLL* para anexar as *DLLs* importantes em processos de sistema, impedindo que *trojans* terminem a execução desses aplicativos.

- Monitorar todas as chamadas do sistema de arquivo e filtrar as operação de tentativa de deleção dos arquivos do módulo.

- Modificar as listas de acesso e permissão do processo protegido, impedindo assim que outros programas consigam terminá-lo.

Essas são apenas algumas maneiras de tentar impedir a desinstalação de um módulo. Mas como estamos tratando aqui de um ambiente do qual não possuímos controle (a máquina do cliente), é quase impossível afirmar que a prevenção da desinstalação será sempre efetiva. Dessa maneira, além de um mecanismo de prevenção de desinstalação, um módulo de segurança deve possuir também um mecanismo de autenticação e validação com o *site* de sua instituição transacional. Assim, no momento em que um usuário acessar o *site*, este deve conseguir verificar, com confiança, que o módulo de segurança necessário não foi desinstalado ou modificado, conforme esquema simplificado apresentado na Figura 5.1.

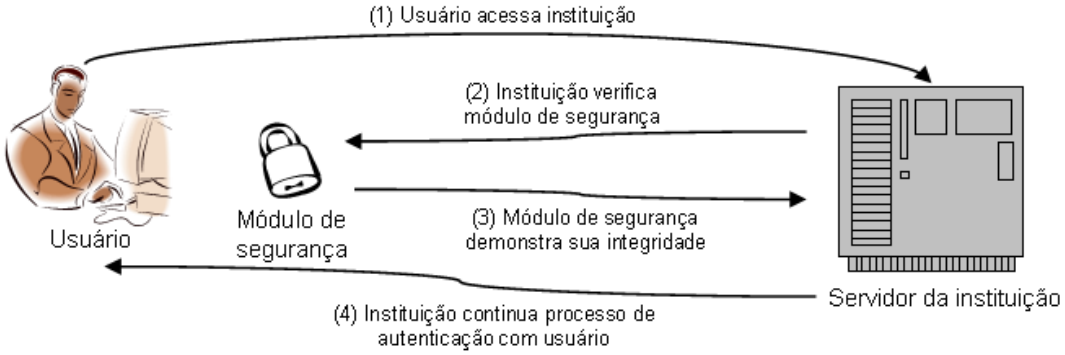

Figura 5.1: Validação do módulo de segurança.

Realizando as validações apresentadas na Figura 5.1, o *site* da instituição terá uma boa garantia de que o módulo de segurança não foi desinstalado nem modificado. Esse método não é à prova de falhas, mas já provê uma boa melhora de segurança.

5.3 Modos de proteção

Nessa seção apresentamos alguns dos métodos que o módulo de segurança pode utilizar para tentar manter um nível de proteção efetivo contra os ataques apresentados nos capítulos anteriores.

5.3.1 Sistemas que operam como antivírus

A proteção do tipo antivírus é talvez umas das mais utilizadas. Essa proteção se baseia no conhecimento prévio dos programas atacantes para realização do bloqueio, ou seja, um antivírus normalmente só bloqueia os vírus que ele conhece. Por um lado, essa proteção pode ser muito efetiva, pois um antivírus recém-atualizado pode bloquear efetivamente vários ataques. Por outro lado, um antivírus está sempre um passo atrás do vírus, sendo necessário o aparecimento primeiro do vírus para depois ser criada a chamada vacina. Além disso, sistemas de antivírus requerem atualizações constantes para manter um nível de efetividade elevado.

Um antivírus funciona a base de assinaturas. Ele verifica dentro de arquivos, possivelmente infectados, a existência dessas assinaturas ou mesmo de um conjunto delas. Assinaturas nada mais são do que cadeias binárias. Caso essas cadeias binárias sejam encontradas dentro de um arquivo, este arquivo é considerado infectado. Um problema que os antivírus possuem é a detecção de falsos positivos. Falsos positivos são programas que possuem alguma assinatura da base de dados do antivírus, mas que na verdade não estão infectados. Assim, o sistema de antivírus detecta erroneamente esse arquivo como infectado. No momento de geração das assinaturas do antivírus, é necessário então tomar cuidado com a detecção excessiva de falsos positivos. Assinaturas muito genéricas, apesar de detectarem uma grande família de vírus, podem vir a trazer muitos falsos positivos. Assinaturas muito específicas podem diminuir muito o número de falsos positivos, mas também não detectam tantos vírus diferentes.

No caso específico de fraudes em instituições que operam sistemas transacionais, não estamos falando exatamente de um antivírus, mas sim de *antitrojan*. Existe uma grande diferença entre esses dois produtos. Vírus normalmente infectam arquivos executáveis, modificando seu comportamento e adulterando seu próprio código binário. Para remover um vírus, é necessário retirar esse pedaço de código binário de dentro do executável. Normalmente esse código binário inserido é fixo, e, portanto, simples de ser encontrado: basta comparar o código inicial com o código adulterado. Por outro lado, a infecção se dá dentro dos próprios arquivos, o que torna necessária a leitura completa de todos os executáveis da máquina para realizar a desinfecção, o que pode ser um processo lento e custoso.

Trojans diferem bastante de vírus. *Trojans* são executáveis separados. *Trojans* têm seu próprio código de execução, e normalmente não modificam nenhum aspecto dos arquivos de sistema. Por um lado, isso os torna simples de serem detectados, pois é necessário apenas verificar quais processos estão em execução no sistema. Por outro lado, *trojans* são incrivelmente parecidos com aplicativos comuns. A maioria dos *trojans* apenas abre algumas janelas e envia algumas informações pela *Internet*. Muitos aplicativos realizam essas tarefas. Portanto, fica difícil encontrar assinaturas que detectem uma grande família de *trojans* e que não apresentem falsos positivos. Como novos *trojans* aparecem a cada dia, assinaturas muito específicas acabam sendo ineficazes. Portanto, é um difícil trabalho achar um meio-termo, encontrando um conjunto de assinaturas efetivas que detecte uma grande gama de *trojans* e ao mesmo tempo não apresente falsos positivos.

Normalmente, um *antitrojan* atualizado faz parte de qualquer módulo de segurança. Nesse caso, a tarefa do antitrojan é proteger o sistema dos *trojans* já conhecidos. Para isso, ele deve ser constantemente atualizado. Isso requer um trabalho de pesquisa e evolução contínua, o que pode se tornar custoso do ponto de vista operacional.

Um sistema de *antitrojan*, dependendo de sua ação, pode ser considerado extre-

mamente intrusivo. Como estamos tratando de clientes, que muitas vezes não têm (e nem desejam ter) conhecimento sobre os aspectos de segurança envolvidos, o sistema de *antitrojan* embutido no módulo de segurança deve ter o mínimo, ou talvez nenhuma interação com o usuário. Ou seja, o sistema deve detectar os *trojans* e realizar a ação necessária sem o conhecimento do usuário.

Esse sistema de *antitrojan*, portanto, estará varrendo todos os programas que estão sendo executados na máquina do cliente. Caso ele encontre algum *trojan* conhecido sendo executado, temos aqui duas opções:

1. Apagar o executável referente ao *trojan* encontrado: esse processo, apesar de extremamente efetivo no aspecto de proteção, pode trazer grandes problemas para a instituição que opere o sistema transacional, principalmente quando se trata da detecção de um falso positivo. Apagar um programa erroneamente da máquina de um cliente pode ser catastrófico.

2. Terminar a execução do *trojan* encontrado: esse procedimento é menos intrusivo, pois estaremos apenas terminando a execução de um processo. Se, por acaso, esse for um falso positivo, o usuário pode sempre executá-lo novamente num tempo futuro. De qualquer maneira, se esse processo encontrado for mesmo um *trojan*, este poderá ser novamente iniciado na próxima inicialização da máquina, sendo assim necessária a constante varredura dos processos em execução para evitar que os *trojans* se reiniciem.

Outro aspecto importante quanto a efetividade e intrusão do sistema *antitrojan* é o momento em que essa proteção deve ser ativada. Nesse caso temos novamente duas alternativas viáveis: iniciar o sistema logo que a máquina for ligada, ou iniciar o sistema somente quando o usuário acessar o *site* da instituição transacional. Cada opção tem suas vantagens e desvantagens. Na primeira opção, a de iniciar o sistema juntamente com a máquina, a vantagem é que a proteção estará sempre ativa. Não existirá

um momento em que um *trojan* poderá atacar e que o sistema não esteja protegido. As desvantagens são duas: primeiro que o sistema estará protegendo o usuário independentemente deste estar acessando o *site* em que necessita de proteção, ou seja, a proteção estará ocorrendo em qualquer *site* que esse usuário possa entrar. Isso nem sempre é uma característica interessante. A outra desvantagem está ligada aos falsos positivos; a finalização de falsos positivos durante o acesso ao *site* transacional é bem menos problemática que a finalização de falsos positivos a qualquer momento. Um falso positivo, nesse caso, teria uma repercussão muito grande e poderia causar sérios problemas à instituição.

A segunda opção, a de ligar a proteção somente no *site* da instituição, tem a vantagem de interferir no funcionamento da máquina do usuário somente quando esse acessa o *site* da instituição, mas por outro lado, deixa o usuário desprotegido nos outros momentos. Um *trojan* poderia explorar essa vulnerabilidade para atacar o sistema.

5.3.2 *Firewall*

Sistemas de *firewall* são extremamente úteis em dois casos: impedir que aplicativos maliciosos se instalem remotamente na máquina do usuário (possivelmente explorando vulnerabilidades do sistema operacional); e impedir que aplicativos maliciosos já instalados localmente consigam enviar os dados capturados para a *Internet*. Apesar disso, sistemas de *firewall* são normalmente complexos de ser operados, não sendo assim muito aconselhados para usuários leigos em informática. Portanto, como estamos aqui tratando de um módulo de segurança que deve ter o mínimo possível de interface com o usuário, o sistema de *firewall* acaba não sendo tão aconselhável para esse cenário. Sistemas de *firewall* mais simples (como por exemplo a *firewall* do *Windows XP Service Pack 2*), acabam sendo úteis apenas para evitar invasões remotas, o que normalmente pode também ser impedido mantendo-se o sistema atualizado com todas as correções de segurança.

Além disso, o nível de intrusão que um sistema de *firewall* pode causar é muito grande. Uma *firewall* deve monitorar todo o tráfego de rede, podendo possivelmente analisar todas as informações que estão sendo transmitidas pela máquina do usuário.

5.3.3 Atualizações de segurança

Uma das maneiras mais rápidas de se espalhar uma nova praga pela Internet é explorando-se vulnerabilidades conhecidas de sistemas operacionais ou aplicativos. Os *worms Sasser* e *Blaster*, citados no capítulo 3 são exemplos disso. Mais recentemente, a vulnerabilidade do formato *WMF* (*Windows Metafile*) (MICROSOFT, 2005a), descoberta em 28 de dezembro de 2005, demorou apenas algumas horas para ser utilizada por atacantes para a disseminação de *worms*, vírus, *spywares* e *trojans* (LABS, 2005). Como muitos usuários não instalam as atualizações necessárias de segurança, essas brechas podem ser exploradas por um bom tempo com grande efetividade.

A instalação dessas atualizações de segurança não é em si uma tarefa que o módulo de segurança possa executar. Essas atualizações devem acontecer utilizando-se as ferramentas dos próprios fabricantes dos sistemas operacionais. Nesse caso, a funcionalidade do módulo de segurança se limita a talvez avisar ao usuário que alguma atualização crítica ainda não foi instalada em seu sistema. Esse ponto, como dito no começo deste capítulo, está mais ligado à conscientização e instrução do usuário sobre navegação segura do que a um aspecto técnico de implementação. De qualquer maneira, a atualização e a correção do ambiente são fatores críticos para a segurança do sistema e devem ser tratadas com cuidado.

5.3.4 *Antikeylogger*

No capítulo 4, descrevemos diversas maneiras que poderiam ser utilizadas para realizar a captura das teclas e dos *cliques* de *mouse* executados pelo usuário. Nesta seção, apresentaremos algumas contramedidas que podem ser adotadas para bloquear

esse tipo de captura.

A utilização de *antikeyloggers* como ferramenta de proteção pode vir a ferir a privacidade do usuário. Como será apresentado a seguir, um *antikeylogger* utiliza a mesma estratégia para bloquear a captura que um *keylogger* utiliza para roubar os dados. O funcionamento dos dois sistemas são muito semelhantes. Portanto, um *antikeylogger* teria a possibilidade de monitorar todas as teclas e *cliques* de *mouse* realizados pelo usuário, inclusive quando este nem estiver acessando o *site* protegido.

5.3.4.1 *Hooks*

A técnica de proteção contra *hooks* é bem semelhante à técnica de captura de teclas utilizando *hooks*. Como explicado no capítulo 4, o *hook* é um mecanismo de interceptação que funciona de maneira encadeada: o sistema operacional, ao receber um tecla, a repassa para o primeiro *hook* da cadeia. Este por sua vez, repassa a mensagem para o segundo *hook* da cadeia, e assim por diante. Ao final, quanto todos os *hooks* registrados já receberam a mensagem, o sistema operacional passa finalmente a mensagem para a janela destino. Esse mecanismo já foi representado na Figura 4.2. Uma possível proteção que o módulo de segurança pode utilizar para prevenir esse método de captura é o seguinte: o módulo obteria um *hook* do sistema operacional. Quando uma mensagem de teclado ou de *mouse* for enviada para esse *hook*, o módulo não repassa essa mensagem para os outros *hooks* instalados, redirecionando-a diretamente para a janela destino. Esse mecanismo está exemplificado na Figura 5.2.

Utilizando esse método, o módulo de proteção consegue efetivamente enganar os outros *hooks* registrados no sistema. Porém, existem alguns problemas inerentes a esse tipo de proteção:

1. O módulo de proteção deve garantir que seu *hook* seja o primeiro na cadeia, conforme representado na Figura 5.2: para realizar essa tarefa, o módulo de

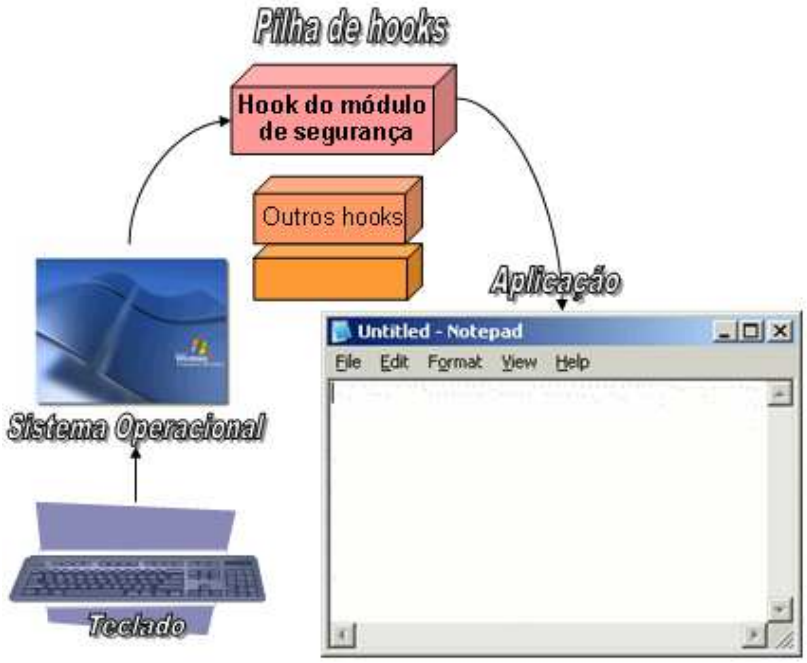

Figura 5.2: Proteção contra captura utilizando *hooks*.

segurança deve voltar constantemente a se registrar como um *hook*. Caso ele faça isso muito lentamente, é possível que algum *trojan* consiga se registrar no meio do caminho e seja então bem-sucedido na captura de algumas teclas; caso ele realize isso muito rapidamente, esse processo pode vir a consumir grandes quantidades de processamento da máquina do usuário, deixando todo o sistema lento.

2. O módulo de proteção estará bloqueando não apenas os *trojans* que tenham instalado *hooks*, mas também todos os outros aplicativos que porventura possam utilizar *hooks*: os *hooks* foram criados como ferramentas para tornar o desenvolvimento de alguns tipos de aplicações mais simples. Alguns exemplos de aplicativos que podem utilizar *hooks* são: aplicações de auxílio a deficientes visuais ou aplicações de leitores de códigos de barras. Nesse caso, essas aplicações passariam a não funcionar corretamente enquanto a proteção de *hooks* estivesse ativa.

Os problemas que acabamos de descrever podem causar grande repercussão e devem ser tratados com cuidado.

5.3.4.2 Captura de campos

Conforme explicado no capítulo 4, a captura de campos de texto é um outro método que um atacante pode utilizar para obter uma senha. Recapitulando rapidamente, esse ataque funciona da seguinte forma:

1. O atacante envia uma mensagem para a janela vítima pedindo para que esta troque seu caractere de senha (que por padrão é o asterisco) para o número 0.
2. O atacante então envia uma mensagem para a janela vítima pedindo o valor do campo de senha.
3. Ao final, o atacante envia uma outra mensagem à janela vítima restaurando o asterisco como o caractere de senha.

A proteção contra esse processo é relativamente simples. Para evitar a captura da senha, tudo que o aplicativo vítima deve fazer é ignorar as mensagens que requisitem seu campo de senha. Apesar de intuitivo, esse comportamento não é padrão para as janelas do *Windows*. Ignorando essa mensagem, o aplicativo evita assim que a captura dos campos de senha seja bem-sucedida.

A solução apresentada, apesar de simples, não é sempre viável. Muitas vezes os aplicativos que estão sendo utilizados para a digitação da senha não estão sob o controle da instituição. Um exemplo muito comum disso é a entrada de senha dentro de um navegador *web*. Esse navegador é desenvolvido por uma empresa terceira e, portanto, não é possível modificar seu comportamento conforme explicado. Nesse caso, será necessário utilizar outro método de proteção para evitar a captura da senha.

Dentro de um navegador *web*, existe uma facilidade chamada *linguagens de script*. Essas linguagens possibilitam um comportamento mais dinâmico das páginas, como a implementação de menus flutuantes ou de validação de formulários. No contexto desta seção, essas linguagens podem ser utilizadas para ajudar a prevenir a captura de campos de senha. Para realizar essa tarefa, o desenvolvedor deve implementar a página de maneira que a senha que será colocada no campo de senha não seja a senha real, mas sim qualquer outra seqüência aleatória de caracteres. A senha real deverá ser armazenada em memória e será colocada dentro do formulário apenas na hora do seu envio ao servidor. Abaixo mostramos passo a passo como seria esse processo de proteção.

1. Usuário digita um caractere da sua senha no campo de senha
2. *Script* da página retira esse caractere do campo de senha e o coloca em uma variável em memória
3. *Script* substitui o caractere no campo de senha por um outro caractere aleatório (um asterisco, por exemplo)
4. Esse processo é repetido até que o usuário digite sua senha completa
5. Ao final, o *script* coloca a senha digitada pelo usuário dentro de um campo escondido e a envia junto ao formulário

Seguindo os passos acima, caso um aplicativo malicioso tente enviar uma mensagem de captura, ele receberá como resposta uma seqüência qualquer de caracteres aleatórios, sendo malsucedido assim na captura da senha.

Em termos da privacidade do usuário, não há nenhum problema em utilizar a proteção acima. Como a senha já estaria sendo digitada na aplicação da instituição, nada foi modificado.

5.3.4.3 *Drivers*

No capítulo 4 apresentamos uma maneira na qual um *driver* poderia ser utilizado para capturar as teclas digitadas pelo usuário. Nesta seção mostraremos que também é possível, de forma muito semelhante, utilizar um *driver* para proteger o sistema contra a captura de teclas.

A idéia da proteção segue a mesma idéia utilizada no sistema de *hooks*. O objetivo aqui é criar um *driver* de baixo nível, do tipo *upper-filter driver*, que possa bloquear todas as teclas digitadas. Esse *driver* passaria as teclas apenas para a aplicação legitima, que então as apresentaria na tela. Essa situação está ilustrada na Figura 5.3.

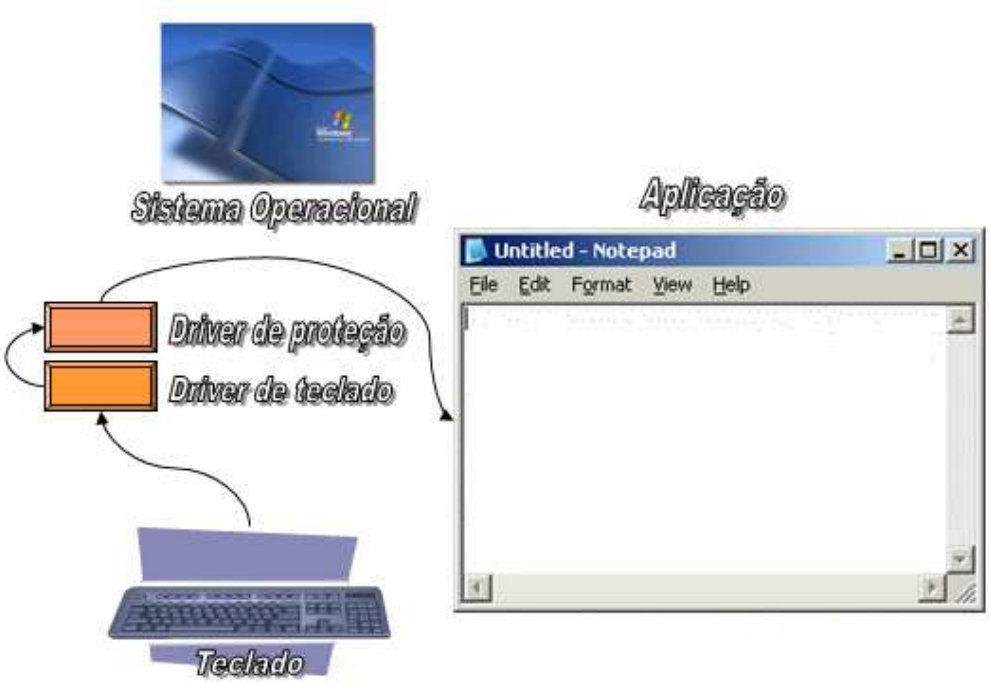

Figura 5.3: Proteção contra captura utilizando *driver*.

Conforme indicado na figura, as teclas digitadas pelo usuário são direcionadas diretamente para a aplicação destino, sem nem mesmo passar pelo sistema operacional.

Essa proteção, apesar de extremamente efetiva, tem diversos problemas envolvidos. Inicialmente, temos o problema do processo de desenvolvimento de um *driver*, que não é muito simples e deve ser feito com muito cuidado, pois qualquer problema

em sua implementação pode causar a queda completa do sistema operacional. Em segundo lugar, temos aqui um problema semelhante ao problema da proteção de *hooks*: outros aplicativos lícitos que estejam na máquina não receberão as teclas digitadas e, portanto, não funcionarão de forma correta. Isso pode ser um grande problema quando tratamos, por exemplo, de aplicativos de ajuda a deficientes visuais que devem capturar todas as teclas para que funcionem corretamente.

O problema mais grave que ocorre na utilização de um *driver* é transmitir as teclas digitadas do *driver* até a aplicação destino de forma segura. Conforme mostrado na Figura 5.3, o *driver* deve repassar diretamente as teclas digitadas para a aplicação final. Mas, de um ponto de vista técnico, a aplicação final não está esperando as teclas vindas do *driver*, e, portanto, não funcionará corretamente. Como conseqüência desse problema, junto com o *driver* é necessário então implementar um aplicativo que execute em modo usuário e que receba as teclas vindas do *driver*. Esse aplicativo, que chamaremos de módulo de suporte, terá então o trabalho de repassar essas teclas para o aplicativo lícito. Deve-se tomar muito cuidado na comunicação entre o *driver* e o módulo de suporte. Um *trojan* poderia tentar explorar essa comunicação para obter as teclas digitadas. É necessário então utilizar métodos criptográficos de autenticação e sigilo dos dados para garantir a confidencialidade dos dados trafegados.

Além disso, dado que o aplicativo de suporte recebeu as teclas corretamente, ainda precisa repassá-las para o aplicativo no qual o usuário está digitando. Esse processo, apesar de simples, pode acabar comprometendo toda a segurança do sistema, pois é nesse ponto que os dados podem ser mais facilmente capturados. É necessário então criar um método seguro para transmitir as teclas entre o aplicativo de suporte e o aplicativo final. Esse pode vir a ser um trabalho muito complexo e extenso.

De qualquer maneira, existem soluções para esse problema. É possível criar uma maneira segura para transmitir os dados entre o aplicativo de suporte e o aplicativo final. As possíveis soluções são extremamente técnicas e fogem do escopo deste tra-

balho, e portanto não serão explicadas aqui.

5.3.4.4 *DLL Injection*

A técnica de *DLL Injection* pode também ser utilizada para prevenir a captura de teclas. A idéia aqui é injetar uma *DLL* dentro dos *trojans* e bloquear o uso das funções do sistema operacional que poderiam ser utilizadas para a captura (como por exemplo as funções de *hook* ou as funções estáticas de teclado).

Como nem sempre é possível identificar qual processo em execução é o *trojan*, esse método acaba tendo que injetar a *DLL* de proteção em todos os processos na máquina. Isso acaba sendo muito perigoso, pois estaremos interferindo nos processos de sistema e poderemos assim estar afetando todo o comportamento da máquina. Além disso, o processo de injeção de *DLL* e de redireciomanento de *API* é extremamente agressivo e pode comprometer a estabilidade da máquina.

De qualquer maneira, esse método pode ser muito eficaz na prevenção da captura de teclas. Mas é necessário cuidado para utilizá-lo da maneira mais correta possível.

No aspecto de privacidade do usuário, a técnica de *DLL Injection* é extremamente intrusiva. Ao injetar uma *DLL* em cada processo do sistema, pode-se em teoria monitorar o comportamento de todas as aplicações sendo executadas. É possível registrar de modo completo todas as interações do usuário com o sistema, clique por clique e tecla por tecla.

5.3.5 *BHO*

A tecnologia chamada *BHO* (*Browser Helper Object*) prove uma maneira simples para que um aplicativo monitore todas as *URLs* navegadas pelo usuário. Para utilizar essa tecnologia, um aplicativo deve simplesmente se registrar no sistema operacional como um *BHO*. Após esse registro, o sistema operacional informará automaticamente

todos os eventos referente à navegação *web* do usuário.

Esse método é muito utilizado por *spywares* para monitoração dos acessos das vítimas. Um *BHO* também tem a funcionalidade de modificar essa navegação, podendo redirecionar o usuário para qualquer endereço desejado. Assim, *spywares* podem verificar o que o usuário está acessando e de tempos em tempos redirecioná-lo para algum *site*, ou mesmo abrir janelas *pop-up* com propagandas.

No contexto de proteção, a tecnologia de BHO pode ser utilizada pelo módulo de segurança para verificar qual endereço o usuário está acessando. Como esse módulo de segurança pertence a uma instituição específica, ele deverá proteger somente as páginas dessa instituição. Assim, utilizando essa tecnologia, o modulo de segurança pode verificar a *URL* acessada e a partir dela decidir se deve ou não realizar a proteção dessa página específica.

A tecnologia de *BHO* é interessante também por outro motivo: o sistema operacional carrega um *BHO* assim que o navegador *web* for aberto. Assim, o módulo de segurança estará em uso desde cedo, dificultando assim a sua remoção (conforme explicado na seção 5.2.2).

Além disso, a tecnologia de *BHO* permite também que aplicativos tenham completo acesso ao conteúdo das páginas navegadas. Um *BHO* pode, por exemplo, verificar todo o código fonte de uma página acessada. Isso pode ser especialmente útil quando estamos lidando com *sites* falsos (acessados possivelmente através de *links* falsos). O módulo de proteção poderia possuir um *BHO* que verifica o código de todas as páginas sendo acessadas. Se o *BHO* percebe que a página atual é muito semelhante à da instituição protegida, mas o endereço navegado pelo usuário não é o da instituição, existe grande chance desse usuário estar acessando uma página falsa.

Em termos de privacidade, as vantagens da tecnologia de *BHO* tornam-se suas piores desvantagens. Um *BHO* pode monitorar completamente a navegação do usuário,

inclusive gravando códigos-fonte das páginas acessadas. Isso torna o *BHO* um espião perfeito, que possivelmente poderia ser utilizado para criar um histórico de navegação do usuário.

5.3.6 Proteção contra redirecionamento

Conforme apresentado na seção 4.4.4, os ataques de redirecionamento têm como objetivo modificar o sistema de resolução de nomes para que o cliente acesse um *site* falso da forma mais transparente possível. Esses ataques se baseiam no comprometimento do servidor de resolução de nome (*DNS*) do usuário ou na modificação dos próprios arquivos de configuração do sistema operacional.

Utilizando a técnica de *BHO* explicada na seção 5.3.5, o módulo de segurança pode realizar a proteção contra o ataque de redirecionamento da seguinte forma:

1. O módulo de segurança se instala como um *BHO*, e passa a monitorar todas as *URLs* navegadas pelo usuário.
2. Ao perceber que o usuário está acessando uma página protegida (a página inicial da instituição protegida, por exemplo), o módulo de segurança realiza uma validação do *IP* para o qual essa *URL* está sendo resolvida.
3. A partir de uma lista de *IPs* pré-cadastrados considerados válidos para essa página, o módulo de segurança verifica se o *IP* obtido acima está nessa lista.
4. Caso o *IP* esteja na lista, ele é considerado válido e o usuário pode continuar com seu acesso.
5. Caso o *IP* não seja encontrado na lista, o módulo de segurança toma alguma providência restritiva.

A providência restritiva tomada pelo módulo de segurança pode variar entre algumas opções, entre elas:

- Redirecionar o usuário para um *IP* considerado correto.
- Colocar uma entrada no arquivo *hosts* da máquina, obrigando o sistema operacional a sempre direcionar uma certa *URL* para um *IP* fixo.

Cada uma das soluções mencionadas têm suas vantagens e desvantagens. A idéia de redirecionar o usuário para um *IP* correto é extremamente transparente para o usuário e não implica em nenhuma modificação no seu padrão normal de navegação. Por outro lado, caso a página acessada seja uma página segura que utilize *SSL*, redirecionar o usuário por *IP* causará a apresentação de um erro quanto ao nome do certificado. Na criação de um certificado *SSL*, é necessário especificar para qual *URL* esse certificado será utilizado. Por exemplo, para um banco de teste, o certificado poderia ser emitido para a *URL www.bancoTeste.com.br*. Quando o usuário acessa essa *URL*, o navegador verifica se o nome digitado bate com o nome do certificado digital. Caso esses nomes sejam diferentes, uma mensagem de aviso é mostrada pedindo a confirmação do acesso. No caso do redirecionamento por *IP*, esse mesmo erro seria apresentado.

A outra solução apresentada, a da colocação de uma entrada no arquivo *hosts* da máquina do usuário, também é extremamente transparente e mantém o padrão normal de navegação do usuário. A desvantagem aqui é que aquela entrada colocada no arquivo ficará lá até que o módulo de segurança a remova. Caso o módulo de segurança seja desinstalado ou pare de funcionar corretamente, é possível que aquela entrada seja esquecida e deixada no arquivo. Com o tempo, caso a instituição mude o *IP* de seu *site*, esse cliente específico não conseguirá mais acessar as páginas dessa instituição. Além disso, existem hoje programas que verificam e protegem o arquivo *hosts* contra modificações. Esses programas poderiam afetar diretamente a efetividade dessa solução.

Existe uma limitação dessa proteção que ocorre quando o usuário utiliza um servidor *proxy*. Um servidor *proxy* é um serviço de rede que permite que clientes façam

conexões de redes indiretas para outros serviços externos. O funcionamento básico de um *proxy* é: o cliente conecta no servidor *proxy* pedindo algum serviço de rede específico (por exemplo, uma página *web*); o servidor proxy realiza a conexão pedida e então repassa os dados de resposta para o cliente.

Em um ambiente utilizando um servidor *proxy*, a resolução de nomes para acessos a páginas web é realizado pelo próprio *proxy*. Ou seja, quando um cliente solicita ao *proxy* a página *www.bancoTeste.com.br*, é o próprio *proxy* que obtém o *IP* dessa *URL* e que realiza a conexão. Portanto, um módulo de segurança instalado localmente na máquina do usuário não consegue saber para qual *IP* o *proxy* resolverá o nome, impedindo assim a proteção local. Nesses casos, a proteção deveria ser colocada no próprio servidor *proxy*, o que normalmente não é uma solução prática.

Em relação à privacidade, as técnicas de proteção contra redirecionamento são muito semelhantes às da tecnologia de *BHO*, na qual o módulo de segurança pode teoricamente analisar todas as *URLs* navegadas pelo usuário.

5.3.7 Detecção de janelas suspeitas

Na seção 4.4.3, mostramos que um dos ataques mais utilizados hoje é o de telas falsas. Nesse ataque, um *trojan* apresenta uma tela semelhante à tela original da instituição atacada, pedindo os dados críticos do usuário, normalmente durante o acesso ao *site* oficial da instituição.

Uma das formas de realizar a proteção contra esse tipo de ataque é a utilização de um *antitrojan*, conforme explicado na seção 5.3.1. Nesse caso, o sistema de *antitrojan* detectaria o *trojan* e finalizaria sua execução antes mesmo de a tela falsa ser apresentada.

O problema da proteção de *antitrojan* é o processo contínuo de atualização das assinaturas. Além disso, a chance de que *trojans* novos (não conhecidos) sejam detec-

tados é relativamente pequena. Portanto, algumas vezes se vê necessária a aplicação de uma proteção extra, que funcione baseada em análise de comportamento em vez de assinaturas de *trojans*.

Para tanto, será necessário analisar qual o comportamento-padrão de uma tela falsa aberta por uma família de *trojans*. Como exemplo, um *trojan* simples que utilize telas falsas normalmente executa os seguintes passos:

1. Verifica que o usuário está acessando uma página específica da instituição a ser atacada
2. Abre uma janela na frente de todas as outras
3. A janela aberta se assemelha muito à da instituição transacional

O módulo de segurança poderia ser configurado para tentar detectar esse comportamento específico e fechar as janelas que forem detectadas por esse método. Nesse exemplo, o módulo de segurança ficaria monitorando o acesso à instituição protegida. Quando esse acesso ocorrer, o módulo verificaria qual janela está atualmente com foco. Se essa janela não for a do navegador, o módulo realizaria uma análise nessa janela (agora considerada suspeita) afim de verificar se ela se assemelha ou não à janela oficial da instituição. Se isso for verdade, essa janela pode ser considerada uma tela falsa e será então fechada.

Apesar da simples explicação, o método apresentado não é de simples execução. O problema mais grave que surge durante o processo de desenvolvimento é o seguinte: como verificar que a janela suspeita se assemelha à janela da instituição?

Pode-se, por exemplo, analisar as palavras que a janela contém e verificar se elas se encontram em uma lista pré-definida de palavras-chave. O problema aqui é que alguns *trojans* podem utilizar imagens ao invés de texto. Outro problema é que janelas

não falsas podem ser facilmente detectadas como janelas suspeitas, resultado em casos de falsos positivos.

É também possível realizar uma análise da estrutura da janela, no sentido de verificar se os campos que a janela da instituição contém também estão na janela que está sendo analisada. Imaginemos novamente o exemplo da tela original do banco apresentada na Figura 5.4 e da tela falsa apresentada na Figura 5.5.

Figura 5.4: Página oficial da instituição.

Podem-se notar diversas semelhanças entre a página original e a tela falsa. Algumas semelhanças seriam: as duas telas possuem três caixas de texto; as três caixas de texto estão alinhadas verticalmente; as duas telas possuem dois campos de entrada de senha; a imagem do cadeado está sempre alinhada à esquerda. E assim por diante. Verificando esse tipo de semelhanças entre as duas janelas, podemos tentar identificar as telas falsas e assim proteger o usuário.

Novamente, deve-se tomar cuidado com o balanceamento entre proteção e facilidade de uso. Se a verificação da estrutura da janela suspeita for muito simples, a proteção será muito eficiente, mas o número de falsos positivos pode ser grande. No outro extremo, se a validação da estrutura da janela suspeita for muito complexa, o

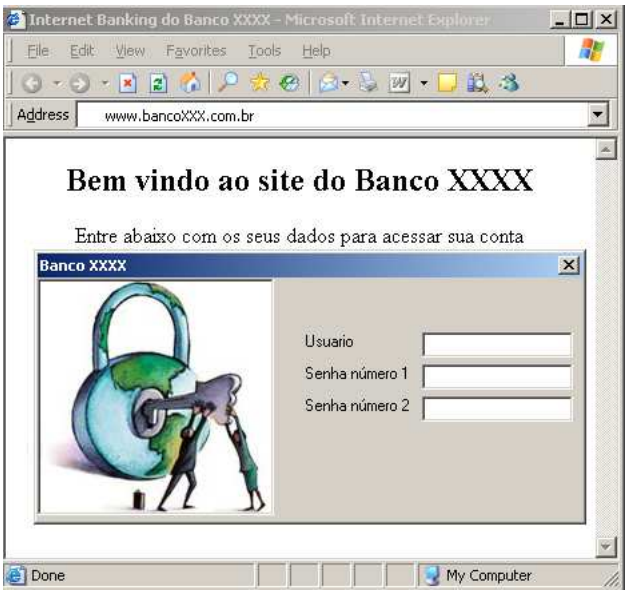

Figura 5.5: Tela falsa atacando o *site* oficial da instituição.

número de falsos positivos deve ser baixo, mas a efetividade da proteção também terá um declínio considerável.

Em termos de privacidade, utilizando essa técnica de verificação de estrutura das janelas, o módulo de segurança poderia teoricamente verificar todos os aplicativos que o usuário acessa e armazenar os textos e imagens contidas em cada um deles. Isso pode ser considerado muito intrusivo pelo usuário final, sendo necessário cuidado ao realizar a implementação dessa técnica.

5.3.8 Proteção da máquina virtual *Java*

O ataque descrito na seção 4.4.5 utiliza a estrutura da máquina virtual *Java* para capturar dados secretos de usuários. No ataque apresentado, as bibliotecas de *run-time* do próprio *Java* eram substituídas, permitindo assim um controle total dos *applets* sendo executados.

O ponto de vulnerabilidade explorado nesse ataque é a falta de validação de integridade das bibliotecas de *run-time*. A proteção contra esse ataque então deve basear-se

neste fato: é necessário criar uma maneira pela qual um programa consiga verificar se a máquina virtual *Java* está íntegra.

Fica claro que nenhum programa *Java* tem a capacidade de verificar a integridade da máquina virtual. Como todos os programas *Java* dependem da própria máquina virtual para executar qualquer função, o atacante pode adulterar essas funções de verificação e assim enganar o programa.

Uma idéia que pode à primeira vista parecer simples e atrativa é a criação de um programa externo, escrito em linguagem nativa da máquina (ou seja, não escrito em *Java*), que verificaria a integridade da máquina virtual. Porém, essa solução tem várias desvantagens: escrevendo um programa em linguagem nativa da máquina, estaremos perdendo a portabilidade do *Java*; existem muitas máquinas virtuais no mercado, e cada uma delas tem diferentes versões e diferentes atualizações, ficando, assim, quase inviável manter uma lista atualizada de todas essas máquinas.

O trabalho de validação da integridade das bibliotecas de *run-time* da máquina virtual *Java* deveria ser passado para os próprios executáveis da máquina virtual. Isso não bloquearia totalmente o ataque, pois os próprios executáveis da máquina virtual poderiam ser trocados. Mas como os executáveis são escritos em código nativo da máquina e têm um tamanho considerável, o ataque se tornaria bem mais complexo. Portanto, uma solução imediata para minimizar o impacto desse ataque seria passar a responsabilidade de validação diretamente para a máquina virtual.

Do ponto de vista de uma instituição que opera um sistema transacional, a solução acima não é facilmente aplicável, pois a máquina virtual *Java* pertence a empresas terceiras, não havendo assim a possibilidade de modificar seu código. Pensando que a instituição só possui um módulo de segurança externo à máquina virtual, hoje ainda não há muito o que fazer. Provavelmente, o mais recomendado seria não utilizar *applets* de autenticação, pelo menos até que as máquinas virtuais passem a verificar a integridade de suas bibliotecas.

5.4 Sugestão de uma solução de curto prazo

Juntando as proteções e recomendações apresentadas anteriormente neste capítulo, podemos agora montar uma solução completa para proteger os clientes de instituições que operam sistemas transacionais da maioria dos ataques aqui apresentados.

Os ataques aqui apresentados foram:

1. *Sites* falsos
2. Exploração de vulnerabilidades
3. *Keyloggers*
4. Telas falsas
5. Ataque de redirecionamento
6. Ataque à máquina virtual *Java*

O único ataque para o qual uma solução direta de curto prazo não foi apresentada é o ataque à máquina virtual *Java*.

Levando em conta as defesas apresentadas neste capítulo, a melhor estratégia para proteção contra cada um desses ataques está apresentada na Tabela 5.1.

Ataque	**Melhor Defesa**
Sites falsos	*BHO*
Exploração de vulnerabilidades	Atualizações de segurança
Keyloggers	*Driver*
Telas falsas	*Antitrojan* + Detecção de janelas suspeitas
Ataque de redirecionamento	*BHO*
Ataque à máquina virtual *Java*	—

Tabela 5.1: Ataques propostos e melhores defesas.

Os motivos para a seleção das proteções expostas já foram indiretamente apresentados durante as discussões deste capítulo, mas iremos colocá-los aqui novamente, agora de maneira mais objetiva e resumida.

A proteção contra *sites* falsos é complicada e exige a análise de cada página que esteja sendo acessada pelo usuário. Para realizar essa tarefa, a tecnologia de *BHO* apresentada pelo sistema operacional é a mais recomendada. Um *BHO* possui interfaces-padrão para acessar o conteúdo da página, e também para redirecionar o usuário para o *site* correto caso ocorra algum problema.

A exploração de vulnerabilidades para instalação local de *software* malicioso também é um problema sério. Um módulo externo de segurança até poderia tentar suprir as deficiências do sistema operacional e se comprometer a proteger essas falhas conhecidas. Mesmo assim, a maneira mais limpa e efetiva de fechar as vulnerabilidades do sistema é a instalação das atualizações de segurança do próprio fabricante do sistema operacional. Nesse caso, o módulo de segurança se limitará talvez a avisar o usuário de que existem atualizações novas ou mesmo baixar e instalar essas atualizações de tempos em tempos.

O ataque utilizando *keyloggers* pode ser extremamente efetivo na captura dos dados críticos dos usuários. A técnica mais efetiva e completa para proteção contra esse ataque é a utilização de *drivers*. A implementação e integração do *driver* com as outras aplicações não é simples, portanto essa técnica exige grandes cuidados no desenvolvimento e nos testes. Mesmo assim, o resultado final de um *driver* bem maduro será o bloqueio da grande maioria dos *keyloggers*.

Já o ataque de telas falsas, o mais empregado atualmente, pode ser reprimido com maior eficiência utilizando-se uma combinação de defesas. A primeira parte da defesa seria um *antitrojan* que conheça as janelas suspeitas já encontradas e analisadas. Essa parte ficaria responsável pela proteção da maioria das telas falsas. Somente algumas telas novas desconhecidas passariam essa barreira. Para essas novas telas falsas, a pro-

teção de detecção de janelas suspeitas baseada na análise comportamental e estrutural da janela seria uma segunda linha de defesa. Para evitar o problema de falsos positivos, essa proteção poderia ser bem restritiva quanto à arquitetura da janela suspeita, já que essa proteção será utilizada somente nos raros casos em que a janela não for conhecida.

O ataque de redirecionamento explicado neste capítulo pode ser melhor combatido utilizando-se novamente a tecnologia de *BHO*. A proteção se basearia numa lista pré-cadastrada de *IPs* válidos para um certo *site*. Caso o *BHO* detecte que o usuário está sendo direcionado para um *IP* não cadastrado na lista, sua navegação seria redirecionada para um dos *IPs* válidos.

Por fim, o ataque à máquina virtual *Java*. Como explicado anteriormente, a melhor proteção contra esse tipo de ataque exige que os próprios desenvolvedores de máquinas virtuais realizem uma validação internamente em seus sistemas. Como isso está fora do poder da instituição, a melhor solução por enquanto seria não utilizar *applets* de autenticação para a entrada de dados críticos dos usuários. De qualquer maneira, as soluções apresentadas para os outros ataques acabam minimizando um pouco o problema da máquina virtual. A utilização de um *antitrojan*, por exemplo, acaba dificultando a instalação de aplicativos maliciosos que possam tentar modificar a máquina *Java*.

Ao final, juntando todas essas proteções citadas, teremos um módulo de segurança bastante eficaz. Mesmo assim, não podemos afirmar que o sistema estará totalmente protegido. Algumas dessas técnicas de proteção tem seus problemas, suas falhas e suas vulnerabilidades. Conforme descrito neste capítulo, nenhuma dessas soluções é perfeita: sempre há alguma brecha que pode ainda ser explorada por atacantes.

É ilusão pensar que a implantação de um módulo de segurança com todas as características citadas nesse trabalho irá resolver o problema das fraudes. Contra os ataques hoje mais difundidos, a solução provavelmente será eficiente. Mas, desde que uma solução seja implantada, a evolução e adaptação dos atacantes ao novo ambiente será rápida: ataques mais modernos e eficientes serão criados em pouco tempo.

Portanto, qualquer solução proposta deve estar associada a um processo contínuo de acompanhamento, investigação e melhorias. Somente operando um processo dessa natureza podemos atenuar significantemente o problema das fraudes em sistemas transacionais.

5.5 A questão da privacidade

Explicamos durante este capítulo os diversos problemas de privacidade que podem vir a ocorrer com as várias técnicas de defesa utilizadas. A privacidade do usuário é seu direito e portanto deve ser respeitada. Contudo, no cenário apresentado, no qual clientes estão utilizando serviços de uma instituição transacional conhecida e confiável, a privacidade toma outra face.

Permitir a instalação de qualquer programa localmente em sua máquina é sempre uma decisão difícil para qualquer usuário. Como garantir que o *software* instalado realize somente aquilo que ele se propõe a fazer?

No cenário de instituições transacionais, já existe uma confiança inerente entre o cliente e a instituição. Esse cliente já utiliza vários serviços dessa instituição para os quais um certo nível de confiança mútua é necessário. Normalmente a instituição já possui vários dados sigilosos desse cliente. O próprio patrimônio financeiro do cliente muitas vezes está nas mãos da instituição. Por esse motivo, o cliente normalmente não se preocupará muito com a instalação em sua máquina de um *software* que possivelmente possa violar sua privacidade, pois ele já confia na instituição e já depende dela para realizar inúmeros serviços.

De qualquer maneira, é interessante seguir algumas boas práticas no desenvolvimento do módulo de segurança a ser instalado. Com isso, o cliente pode aferir, com uma confiança razoável, que o *software* instalado localmente não está violando sua privacidade. Abaixo seguem algumas dessas boas práticas que deveriam ser seguidas

para manter a menor intrusão possível na máquina do cliente.

- O módulo de segurança só deve estar ativo durante o acesso às páginas da instituição: quando o cliente acessa explicitamente, por livre e espontânea vontade, as páginas da instituição, só então a proteção deve ser ligada. É somente nesse momento que o módulo poderá verificar a existência de *trojans* ou programas maliciosos em geral. Por um lado, isso acaba afetando um pouco a segurança do sistema, pois seria mais eficiente se o módulo de segurança pudesse estar ativo desde a inicialização da máquina. Mesmo assim, a perda de segurança envolvida é pequena quando comparada à grande vantagem que se tem em termos de privacidade: o usuário não poderá afirmar que o módulo está capturando suas informações de outras instituições ou que está afetando o desempenho da máquina fora do acesso à instituição protegida. Toda a atividade do módulo fica limitada ao tempo em que o cliente está dentro da instituição. E como o foi o próprio usuário que, por livre arbítrio, digitou no navegador o endereço protegido, fica também sobre ele o comando da ativação e desativação da proteção.

- O módulo de segurança deve se comunicar apenas com o servidor da instituição protegida: para evitar problemas de acusações de envio de informações para destinatários ilícitos, o módulo de segurança deve enviar e receber informações apenas do próprio *site* da instituição protegida.

- O módulo de segurança deve trocar o mínimo de informação possível com o servidor da instituição: os dados trocados entre os dois lados devem se limitar à baixa de atualizações do módulo e ao envio de incidentes identificados na máquina do cliente. Sendo assim, os dados trafegados devem ser bem escassos. Dessa maneira, um cliente poderá verificar que o módulo não está enviando informações que violem sua privacidade, pois a banda utilizada pelo mesmo é muito pequena para realizar tal tarefa.

- O módulo de segurança deve possuir um modo simples de desinstalação: um cliente sempre deve ter o direito de escolher se quer ou não instalar um *software* localmente em sua máquina. Caso o cliente decida que quer desinstalar o módulo de segurança, deve haver uma maneira simples e prática de realizar essa tarefa. Obviamente, esse método de desinstalação deve ser bem estruturado para que *trojans* não possam usa-lo como um ponto de vulnerabilidade do sistema.

Seguindo essas premissas de funcionamento, a instituição terá mais força no momento de afirmar que seu módulo de segurança não viola a privacidade de seus clientes. Assim é possível obter um compromisso razoável entre segurança e privacidade.

6 CONSIDERAÇÕES SOBRE SOLUÇÕES ESTRUTURAIS

"The human factor is truly security's weakest link"

Kevin Mitnick

Durante o decorrer deste trabalho, foram explicadas técnicas de ataque e defesa que visam impedir as fraudes virtuais. Todo essa batalha se deve a um único fator: a falta de confiança na máquina do cliente. É por esse simples motivo que o módulo de segurança criado no capítulo 5 é necessário. É explorando quase que unicamente esse fator que os atacantes conseguem realizar os ataques apresentados no capítulo 4. Ou seja, tudo o que foi apresentado aqui gira em torno do ambiente promíscuo e não confiável que é a máquina do cliente.

Portanto, para que se consiga acabar (ou ao menos diminuir consideravelmente) com as fraudes virtuais, é necessário lutar contra esse fator importante. Em termos bem genéricos, existem duas maneiras de se lidar com o problema da não confiança na máquina do cliente:

1. Realizar o acesso ao *Internet Banking* (ou ao menos o processo de autenticação) a partir de outro ambiente confiável.

2. Tornar de alguma maneira a máquina do cliente novamente confiável.

A primeira solução parece ser a mais diretamente acessível, pois podemos utilizar

artifícios tecnológicos para realizá-la. Alguns exemplos interessantes de ambientes confiáveis seriam:

- Utilizar dispositivos externos dedicados, sem comunicação com o exterior, para realizar o processo de autenticação e de assinatura digital de cada transação efetuada.
- Utilizar um sistema operacional secundário, também dedicado, para acessar o serviço de *Internet Banking* (exemplo: utilizar um disco de *boot* totalmente autosuficiente, não alterável, que somente permite que o usuário utilize o serviço provido pela instituição).
- Utilizar um sistema de máquinas virtuais totalmente segregado, em que a máquina virtual utilizada para acesso ao *Internet Banking* esteja isolada e possa realizar somente essa tarefa.

Esses são alguns exemplos práticos de como se poderia utilizar ambientes confiáveis para a redução das fraudes. Note que nenhum desses métodos é prático do ponto de vista do cliente; há sempre um processo custoso a ser realizado pelo cliente antes do seu acesso ao serviço. Além disso, alguns desses exemplos citados têm problemas de logística e custo, pois não é simples nem barato distribuir dispositivos dedicados ou discos de *boot* para milhares ou mesmo milhões de clientes. De qualquer maneira, em termos de segurança, os exemplos acima mantêm sua efetividade independentemente do estado da máquina do cliente.

A outra alternativa para lidar com o problema das fraudes seria tornar a máquina do cliente novamente confiável. Nesse ponto, encontramos um problema de educação a ser tratado. As máquinas dos clientes hoje estão infectadas pela falta de informação e ingenuidade das pessoas. Seria então necessário um grande processo de educação e conscientização dos usuário de serviços de *Internet Banking* para que esses passem a

se preocupar mais com sua própria segurança. Esses usuários deveriam ter conceitos sobre navegação segura, sobre certificação digital, sobre utilização de aplicativos de segurança, entre outros. Como muitos dos usuários desses serviços são leigos em informática, a tarefa de educá-los em segurança não é simples. Apesar disso, esse parece ser um mal necessário, pois a segurança sempre, em algum ponto, dependerá da pessoa que está utilizando o sistema, por mais seguro que ele seja.

Parece que uma das alternativas mais eficazes para promover a defesa contra todos esses ataques seja a criação nativa (direto de fábrica) na máquina do cliente de múltiplas máquinas virtuais, sendo que uma delas seria destinada para execução e navegação de aplicações sensíveis que exigem alta segurança.

O ambiente normal de uso da máquina do cliente seria totalmente dissociado e completamente isolado desse ambiente de segurança. A própria implementação do monitor de máquinas virtuais seria simplificada para que ele tivesse pouquíssimo código, muito menor que os 50 milhões de linhas de código do *Windows* para minimizar a probabilidade de existência de vulnerabilidades.

Infelizmente, a aplicabilidade desse tipo de solução depende dos fornecedores de máquinas e sistemas operacionais. Aparentemente, certos indícios (MICROSOFT, 2005b) indicam que eles estão trabalhando nessa direção. Se uma solução desse tipo não for logo disponibilizada, a utilização de computadores pessoais para a execução de transações sensíveis ficará bastante limitada.

Enquanto isso não ocorre, a solução apresentada neste trabalho, complementada por um processo contínuo de melhorias, é uma das alternativas viabilizadoras do uso de computadores pessoais de baixo custo executando transações sensíveis com um nível de segurança adequado.

7 CONCLUSÕES

"...what is proved by impossibility proofs is lack of imagination."

John Bell

Apresentamos neste trabalho uma perspectiva da situação atual das fraudes virtuais contra instituições que operam sistemas transacionais no Brasil. Foi mostrado que o problema das fraudes já é muito sério e ainda tende a crescer. Foram apresentados os ataques mais utilizados hoje para realizar essas fraudes, e para cada um desses ataques foi explicada uma possível forma de proteção. Demonstrou-se também que, com um conjunto dessas defesas, pode-se criar um sistema eficiente de combate às fraudes. Mesmo assim, uma solução definitiva para as fraudes requer ainda um trabalho mais árduo, envolvendo ação que deve ser executada preferencialmente pelos fornecedores (fabricantes) de máquinas e sistemas operacionais.

A solução de curto prazo aqui apresentada contra as fraudes obviamente não provê 100% de eficácia. Após a leitura deste trabalho, deve ficar claro que não existe uma solução puramente tecnológica que resolverá esse problema. É necessário aplicar, em conjunto com a tecnologia, um processo de educação e criação de um cultura de segurança nos usuários dos serviços atacados.

A técnica conhecida como "engenharia social" pode quase sempre sobrepassar o mais seguro dos sistemas. A ignorância e inocência das pessoas continuarão a ser os grandes pontos de vulnerabilidade de qualquer sistema.

As técnicas aqui descritas podem ajudar, e muito, uma instituição transacional na batalha contra as fraudes. Mas a solução definitiva só virá com a conscientização e o compartilhamento da responsabilidade da segurança do sistema com seus usuários.

REFERÊNCIAS

AARONSON, L. For love of money - malicious hacking takes an ominous turn. *Spectrum, IEEE*, v. 42, p. 17 – 19, Nov. 2005.

AWAD, K. F. N. F. The deceptive behaviors that offend us most about spyware. *Communications of the ACM*, v. 48, p. 55 – 60, Aug. 2005.

BERTIN, M. The new security threats. *Ziff Davis Smart Business*, Feb. 2001.

BRUSTOLONI, X.; BRUSTOLONI, J. C. Hardening web browsers against man-in-the-middle and eavesdropping attacks. *International World Wide Web Conference*, p. 489 – 498, 2005.

CASS, S. Anatomy of malice [computer viruses]. *Spectrum, IEEE*, v. 38, p. 56 – 60, Nov. 2001.

CERT. *CERT/CC Statistics 1988-2005*. 2005. Hipertexto. Disponível em: http://www.cert.org/. Acesso em: 16 de maio de 2006.

COHEN, F. 50 ways to attack your web systems. *Internet Holes*, Jan. 1995.

DEAN E. W. FELTEN, D. S. W. D.; BALFANZ, D. Java security: Web browsers and beyond. *Internet Beseiged: Countering Cyberspace Scofflaws*, October 1997.

DESCRIPTION, A. R. M. *Anti-Spyware Coalition Risk Model Description*. 2005. Hipertexto. Disponível em: http://www.antispywarecoalition.org/documents/RiskModelDescription.htm. Acesso em: 16 de maio de 2006.

DHAMIJA, L. D. T. R. The battle against phisihng: Dynamic security skins. *Proceedings of the 2005 ACM Symposium on Usable privacy and security*, p. 77 – 88, July 2005.

FOLHA. *Folha Online*. 2005. Hipertexto. Disponível em: http://www.folha.uol.com.br/. Acesso em: 16 de maio de 2006.

FREEDMAN, D. H. *How To Hack A Bank*. 2000. Hipertexto. Disponível em: http://www.forbes.com/asap/2000/0403/056.html. Acesso em: 16 de maio de 2006.

FREEMAN, A. U. L. A. Why do people hate spyware? *Communications of the ACM*, v. 48, p. 50 – 53, Aug. 2005.

GIBSON, S. Spyware was inevitable. *Communications of the ACM*, v. 48, p. 37– 39, Aug. 2005.

GOLDBERG D. WAGNER, R. T. I.; BREWER, E. A. A secure environment for untrusted helper applications: Confining the wily hacker. *Proceedings of the 6th Usenix Security Symposium*, 1996.

HOWARD, J. D. *An Analysis of Security Incidents on the Internet*. Tese (Doutorado) — Pittsburgh, Pennsylvania, 1997.

HU, T. D. Q. Is spyware an internet nuisance or public menace? *Communications of the ACM*, v. 48, p. 61 – 66, Aug. 2005.

HUBERMAN B.A. ADAR, E. F. L. Valuating privacy. *Security & Privacy Magazine, IEEE*, v. 3, p. 22– 25, Sept.-Oct. 2005.

HYDE, R. *The Art of Assembly Language Programming*. 1. ed. [S.l.]: No Starch Press, 2003. http://courses.ece.uiuc.edu/ece390/books/artofasm/artofasm.html.

IDGNOW. *International Data Group (Brasil)*. 2005. Hipertexto. Disponível em: http://www.idgnow.com.br/. Acesso em: 16 de maio de 2006.

KOSKOSAS, I. V.; PAUL, R. J. The interrelationship and effect of culture and risk communication in setting internet banking security goals. *Proceedings of the 6th international conference on Electronic commerce*, v. 60, p. 341 – 350, 2004.

LABS, K. *Trojan programs exploiting the latest Windows vulnerability*. 2005. Hipertexto. Disponível em: http://www.kaspersky.com/news?id=176708581. Acesso em: 16 de maio de 2006.

LANDWEHR ALAN R. BULL, J. P. M. C. E.; CHOI, W. S. A taxonomy of computer program security flaws. *ACM Computing Surveys (CSUR)*, v. 26, n. 3, p. 211 – 254, 1994.

LEE, K. A. K. Y. Investigating factors affecting the adoption of anti-spyware systems. *Communications of the ACM*, v. 48, p. 72 – 77, Aug. 2005.

LOBL, T. R. Student papers: Identity theft, spyware and the law. *Proceeding of the 2nd annual conference on Information security curriculum deevlopment InfoSecCD '05*, Sep. 2005.

MADSEN YUZO KOGA, K. T. P. Dim framework: Federated identity management for protecting users from id theft. *Proceeding of the 2005 workshop on Digital identity management*, Nov. 2005.

MATAYOSHI, C. M. Modelo de segurança da linguagem java. *SBRC*, 1998.

MCGRAW, G.; FELTEN, E. *Securing Java*. [S.l.]: John Wiley & Sons, Inc., 1999. http://www.securingjava.com/.

MICROSOFT. *Microsoft Security Bulletin MS06-001*. 2005. Hipertexto. Disponível em: http://www.microsoft.com/technet/security/bulletin/ms06-001.mspx. Acesso em: 16 de maio de 2006.

____. *Next-Generation Secure Computing Base*. 2005. Hipertexto. Disponível em: http://www.microsoft.com/resources/ngscb/default.mspx. Acesso em: 16 de maio de 2006.

MITTELSDORF, A. W. *Uma plataforma para computação com confiança baseada em monitor de máquinas virtuais e atestamento dinâmico*. Tese (Doutorado) — Escola Politécnica da Universidade de São Paulo, São Paulo, 2004.

MOORE VERN PAXSON, S. S. C. S. S. S. D.; WEAVER, N. *The Spread of the Sapphire/Slammer Worm*. 2003. Hipertexto. Disponível em: http://www.caida.org/outreach/papers/2003/sapphire/sapphire.html. Acesso em: 16 de maio de 2006.

MSDN. *Microsoft Corporation - Hooks*. 2005. Hipertexto. Disponível em: http://msdn.microsoft.com/library/default.asp?url=/library/en-us/winui/winui/windowsuserinterface/windowing/hooks.asp.Acesso em: 16 de maio de 2006.

NILSSON ANNE ADAMS, S. H. M. Building security and trust in online banking. *Conference on Human Factors in Computing Systems*, p. 1701 – 1704, 2005.

POSTON THOMAS F. STAFFORD, A. H. R. Spyware: a view from the (online) street. *Communications of the ACM*, v. 48, Aug. 2005.

PUENTE F. GONZALEZ, S. S. J. de la. Virus attack to the pc bank. *Security Technology, 1999. Proceedings.*, p. 304 – 310, Oct. 1999.

PUENTE J.D. SANDOVAL, P. H. F. de la. Personal digital signer for internet banking. *Communications, Computers and signal Processing, 2003. PACRIM*, v. 2, p. 700 – 703, Aug. 2003.

PUENTE JUAN D. SANDOVAL, P. H. F. D. L. Pocket device for authentication and data integrity on internet banking applications. *Security Technology, 2003. Proceedings*, p. 43 – 50, Oct. 2003.

PUENTE SANTIAGO GONZALEZ, J. D. S. Fernando de la; HERNANDEZ, P. Viral attack to internet banking applications. *Aerospace and Electronic Systems Magazine, IEEE*, v. 15, p. 3 – 8, 2000.

SCHMIDT, K. P. A. M. B. Spyware: a little knowledge is a wonderful thing. *Communications of the ACM*, v. 48, p. 67 – 70, Aug. 2005.

SEGEV, J. P. A.; ROLDAN, M. Internet security and the case of bank of america. *Communications of the ACM*, v. 41, n. 10, p. 81 – 87, October 1998.

SLEWE, T.; HOOGENBOOM, M. Who will rob you on the digital highway? *Communications of the ACM*, v. 47, n. 5, p. 56 – 60, May 2004.

SMARTLINK. *Virtual Keabord Example*. 2005. Hipertexto. Disponível em: http://www.smartlinkcorp.com/. Acesso em: 16 de maio de 2006.

SMITH, S. Pretending that systems are secure. *Security & Privacy Magazine, IEEE*, v. 3, p. 73 – 76, Nov.-Dec. 2005.

SNADBOY. *SnadBoy's Revelation*. 2000. Hipertexto. Disponível em: http://www.snadboy.com/. Acesso em: 16 de maio de 2006.

STYTZ, M. Protecting personal privacy: Hauling down the jolly roger. *Security & Privacy Magazine, IEEE*, v. 3, p. 72– 74, July-Aug. 2005.

THOMPSON, R. Why spyware poses multiple threads to security. *Communications of the ACM*, v. 48, p. 41 – 43, Aug. 2005.

TIMES eCommerce. *Identity Theft Is Fastest-Growing Online Crime*. 2003. Hipertexto. Disponível em: http://www.epaynews.com/index.cgi?survey=&keywords=ID%20theft&optional=&subject=&location=&ref=keyword&f=view&id=10564533976222152l2&block=1. Acesso em: 16 de maio de 2006.

VENNERS, B. Java's security architecture - an overview of the jvm's security model and a look at its built-in safety features. *Java World*, August 1996.

____. Security and the class loader architecture - a look at the role played by class loader in the jvm's overall security model. *Java World*, September 1997.

____. Security and the class verifier - a look at the role played by class verifier in the jvm's overall security model. *Java World*, October 1997.

VLIET, H. van. *Mocha, the Java Decompiler*. 1996. Hipertexto. Disponível em: http://www.brouhaha.com/~eric/software/mocha/.Acesso em: 16 de maio de 2006.

WARKENTIN XIN LUO, G. F. T. M. A framework for spyware assessment. *Communications of the ACM*, v. 48, p. 79 – 84, Aug. 2005.

WENUIN GUANGLIN HUANG, L. X. Z. M. X. D. L. Detection of phishing webpages based on visual similarity. *Special interest tracks and posters of the 14th international conference on World Wide Web*, May 2005.

WHEELER, A. C. D.; LUO, A. X. J. Java security extensions for a java server in a hostile environment. *17th Annual Computer Security Applications Conference (ACSAC'01)*, December 2001.

ZHANG, X. What do consumers really know about spyware? *Communications of the ACM*, v. 48, p. 44 – 48, Aug. 2005.

Apêndice A – EXEMPLO DE *E-MAILS*

Figura A.1: Exemplo de *e-mail* contendo *trojan*.

Figura A.2: Exemplo de *e-mail* contendo *trojan*.

Figura A.3: Exemplo de *e-mail* contendo *trojan*.

Ministério da Justiça

Departamento de Polícia Federal

Invasores!

Os **hacker's**: deixam de ser personagens de ficção e se espalham pelo planeta.

Fabrício Rocha
Da equipe do Correio

Os Bancos são protegidos.

A Polícia Federal prendeu 54 pessoas de uma quadrilha de hackers (piratas da internet) que desviou R$ 80 milhões de contas correntes de diversos bancos. As prisões foram feitas em quatro estados: 26 no Pará; 12 no Ceará; nove em Tocantins; e seis no Maranhão. Os agentes federais saíram para cumprir 81 mandados de prisão. Entre os detidos, 18 eram reincidentes no crime.

A quadrilha foi investigada durante quatro meses através de escutas telefônicas e acompanhamento dos acessos à internet feitos pelos principais integrantes do grupo. Os hackers enviavam e-mails infectados com um vírus que capturava a senha e os dados sigilosos dos clientes de banco. Por dia chegavam a ser transmitidas mais de 30 mil mensagens eletrônicas para todo o país.

Veja aqui as fotos dos principais integrantes...

Figura A.4: Exemplo de *e-mail* contendo *trojan*.

GLOSSÁRIO

- Engenharia Social: prática para obtenção de informações confidenciais através da manipulação de usuários legítimos. O atacante normalmente utilizará um telefone ou a *Internet* para enganar as pessoas e fazer com que essas revelem informações sensíveis ou realizem alguma ação que vai contra sua política de segurança. Em vez de explorar falhas de segurança do sistema, esse método explora a tendência natural das pessoas de confiar na palavra alheia.

- *Malware*: programa com o objetivo de infiltrar ou danificar um computador, sem o consentimento de seu dono. Inclui as classes *trojans*, *worms*, *spywares* e vírus.

- *Spyware*: ampla categoria de programas maliciosos criados para interceptar ou tomar controle parcial das operações de um computador sem o consenso do usuário legítimo. Táticas comuns utilizadas por *spywares* são: mostrar janelas de propaganda não solicitadas; roubo de informação pessoal (como números de cartão de crédito); monitoração das atividades de navegação na *Internet* para fins de *marketing*; redirecionamento de requisições *web* para páginas com propagandas.

- *Trojan*: programa malicioso que se apresenta disfarçado como um programa legítimo. *Trojans* não possuem a habilidade de se replicar automaticamente, normalmente dr utilizando da curiosidade ou inocência da vítima para serem executados.

- Vírus: programas que se replicam automaticamente e se espalham através da inserção de cópias de si mesmo dentro de outros executáveis ou documentos.

- *Worm*: programa que se replica automaticamente, similarmente a um vírus. En-

quanto um vírus deve se anexar a (ou se tornar parte de) um outro executável, os *worms* são totalmente autocontidos e não necessitam de outros programas para se propagar.