UNIVERSIDADE FEDERAL DA PARAÍBA

CENTRO DE CIÊNCIAS E TECNOLOGIA

COORDENAÇÃO DE PÓS-GRADUAÇÃO EM INFORMÁTICA

ESTUDO E VERIFICAÇÃO DA SEGURANÇA EM *SITES* INTERNET

Claudio de Castro Monteiro

Campina Grande - Pb

Dezembro – 1997

Claudio de Castro Monteiro

ESTUDO E VERIFICAÇÃO DA SEGURANÇA EM *SITES* INTERNET

Dissertação apresentada ao Curso de MESTRADO EM INFORMÁTICA da Universidade Federal da Paraíba, em cumprimento às exigências para a obtenção do grau de Mestre.

Área de Concentração: **Redes de Computadores e Sistemas Distribuídos**

Francisco Vilar Brasileiro
(Orientador)

Campina Grande - PB

Dezembro - 1997

ESTUDO E VERIFICAÇÃO DA SEGURANÇA EM SITES INTERNET

CLÁUDIO DE CASTRO MONTEIRO

DISSERTAÇÃO APROVADA EM 19.12.1997

Francisco Vilar Brasileiro

PROF. FRANCISCO VILAR BRASILEIRO, Ph.D
Presidente

PROF. JOSÉ ANTÃO BELTRAO MOURA, Ph.D
Examinador

PROF. PEDRO SÉRGIO NICOLLETTI, M.Sc
Examinador

CAMPINA GRANDE - PB

A meu pai, minha mãe e meu irmão.

"Um homem sem humildade não
passa de um aglomerado de
informações e experiências."

CCM

AGRADECIMENTOS

É um risco muito grande tornar pessoais os agradecimentos de um trabalho que envolveu uma grande e árdua jornada que começou em 1991, quando iniciei meu mestrado. Citar nomes poderia ser uma grande injustiça, pois muitas pessoas contribuíram decisivamente para que este trabalho, que é mais do que um resultado simples de uma pesquisa, pudesse ser realizado.

Comecei meu curso em 1991 e no ano seguinte fui obrigado a interrompê-lo por quase 5 anos, retornando em 1997. Durante esse período não posso deixar de agradecer a meu pai, minha mãe e meu irmão que sempre me apoiaram para que pudesse chegar ao final deste trabalho.

Agradeço também à Carolina, pessoa ímpar e fundamental em minha vida, com a qual aprendi conceitos impossíveis de serem esquecidos e que dispensou a mim grande parte de seu carinho, amor e amizade.

Meus sinceros agradecimentos ao grande amigo Denis Araújo, pela inestimável ajuda durante toda minha estadia em Campina Grande, com quem aprendi lições técnicas e de vida que jamais poderei esquecer.

A Zelito e Idália, pela força e apoio a mim dispensados durante a realização deste trabalho.

A José Gomes, Lucia, Lucilene, Marcio, Marcia, Luiz Augusto e Michele pela imensa paciência e amizade a mim dispensadas, sem as quais não teria obtido sucesso no decorrer de meu trabalho.

A Edjander, Edjair, Frazão, Soninha, Érico, Fernandinho, Araripe, Walfredo, Tojal, amigos de ontem, de hoje e de sempre que, embora tenham seguido seus caminhos, sempre estarão em minha mente e em meu coração, pelo tempo de convivência e pelas batalhas superadas.

Aos amigos Flavius, Fábio, Claudio, Josenilda, Vera, Lilian, Érica, Vladimir, Elissandro, Gustavo, Fellipe, Guga, Luiza, Mila, Michele, Ricardo, Cristiane, Tércio, Raquel, Flavia, Raissa, Karina, Livia, Rodrigo, pela forma como me receberam no DSC e no LSD e pela ajuda em momentos difíceis vividos durante a última etapa de minha pesquisa.

Ao professor e amigo Francisco Vilar Brasileiro (Fubica), pela confiança que depositou em mim e em minha proposta de trabalho, mostrando-se sempre presente com suas críticas sempre construtivas e opiniões impessoais, orientando o trabalho rumo ao fina' bem sucedido.

A todo o pessoal da COPIN e do DSC que não economizam esforços no sentido de auxiliar nós alunos e que, acima de tudo são amigos de verdade.

A Walkíria, pessoa ímpar que se tornou mais, muito mais do que o grande amor da minha vida e a quem devo muito de minha vida aqui em Campina Grande. Walkíria, um obrigado é muito pouco. TE AMO!!!

Enfim, agradeco a todos que comigo compartilharam dos momentos difíceis e agradáveis dessa jornada, esperando que os laços de amizade aqui formados sejam constituídos de fios fortes de humildade, compreensão, carinho e apreço, visando tornar eterno esse sentimento.

Muito obrigado a todos....

Resumo

O grande desenvolvimento da Internet nos últimos anos tem gerado uma crescente demanda por profissionais que administrem *sites* com eficiência e garantindo a segurança da informação armazenada. Ao mesmo tempo tem-se observado no Brasil um número crescente de ataques a esses *sites*. Isso sugere a existência de dois problemas: i) a oferta ou a disponibilidade de profissionais competentes para desempenhar a tarefa de administração desses *sites* é insuficiente; e ii) as ferramentas utilizadas para garantir a segurança desses *sites* não são satisfatórias. Nesse trabalho apresentamos o ambiente Groove que é composto por um conjunto de ferramentas que podem ser usadas tanto para capacitar os administradores de *sites* ligados à Internet, como para monitorar e corrigir possíveis falhas de segurança existentes nesses *sites*. Diferentemente de outras ferramentas específicas disponíveis, o ambiente Groove integra uma gama extensa de serviços, de fácil utilização e com atualização contínua.

Abstract

The Internet has experienced a fantastic growth in the past few years. This fact has generated an enormous demand for professionals with the ability to manage sites efficiently and providing the required level of security for users' and system's data. In Brazil, particularly, it has been observed an increasing number of attacks to sites connected to the Internet, which suggest two problems: i) the lack or unavailability of well qualified site managers; and ii) the lack of efficient tools for guaranteeing security. In this paper we introduce the Groove environment, composed by a set of tools that can be used both for training site managers, as well as monitoring the behaviour of a site, indicating possible vulnerabilities. Unlike other specific tools available, the Groove environment integrates a comprehensive set of services that can be easily used and that are continuously upgraded.

Índice

Lista de Figuras

Capítulo 1
Introdução

1.1. Importância

Hoje, sabemos que o "poder" da *Internet* é indiscutível. Temos observado que ao mesmo tempo que cresce a demanda de serviços *Internet*, cresce junto com ela a oferta de acessos, providos por instituições públicas e privadas, tendo essas últimas encontrado seu mercado no Brasil a partir de maio de 1995 com a abertura comercial da rede mundial de computadores.

Juntamente com o aumento da oferta de acesso, vem crescendo também outra demanda: a de profissionais treinados para desenvolver as funções de administração dos *sites*[1] que surgem a todo dia. Esses profissionais devem ser capazes de, dentre outras coisas:

a) Instalar, configurar e disponibilizar serviços eficientes e eficazes para seus usuários;

b) Prover mecanismos que possam medir e aumentar o nível de segurança de seus *sites*, visando a integridade de seus dados e dos dados de seus usuários.

Dentro desse contexto, o que se tem visto é uma carência desse profissional, gerando um grande número de *sites* vulneráveis a ataques de "visitantes" indesejados. Esses visitantes usam vários artifícios para

[1] Site: Ambiente computacional onde são armazenadas e tratadas informações que trafegam na Internet

fundamentar seus conhecimentos sobre os *sites* a atacar. Não surpreendentemente, um dos artifícios mais usados são as listas de discussão públicas frequentadas por administradores de *sites,* que ingenuamente expõem suas dúvidas, fornecendo "pistas" sobre as possíveis vulnerabilidades em seus sistemas.

Sensível a este quadro preocupante, essa pesquisa visa sistematizar o conhecimento sobre as vulnerabilidades mais comuns em *sites Internet* e sobre os tipos de ataques mais frequentes, propondo sua classificação. Essa classificação busca os seguintes objetivos:

a) Organizar o conhecimento relativo às vulnerabilidades existentes nos sistemas operacionais que suportam *sites Internet* e nas aplicações comumente disponíveis nesses *sites*;

b) Mostrar como as principais aplicações de sistemas operacionais e do conjunto de protocolos TCP/IP estão vulneráveis, e dessa forma gerar subsídios que possibilitem os desenvolvedores de aplicações evitar ou pelo menos limitar vulnerabilidades em novas aplicações desenvolvidas.

Sabemos que, em sistemas baseados em TCP/IP, o aspecto de segurança vem sendo tratado com sucesso por mecanismos de *firewall*[2]. No entanto, esses mecanismos possuem um nível de robustez e complexidade proporcionais ao valor do investimento na tecnologia. Além disso, quanto mais completa for a solução *firewall*, mais limitações serão impostas aos usuários dos serviços do *site*. Isso faz com que poucos sites utilizem essa tecnologia, ou ainda, mesmo que essa tecnologia seja usada, nem sempre todas as vulnerabilidades são resolvidas, quer por falhas na configuração do

[2] Firewall: Metodologia utilizada para proteger uma rede de computadores de informações não desejadas provenientes de outras redes ligadas à Internet

firewall, quer por decisões que visem evitar uma redução na usabilidade dos serviços, quer por limitações da própria tecnologia de firewall.

Esse trabalho visa também a especificação e implementação de uma ferramenta de verificação do nível de segurança de um dado *site*, seguindo o modelo de classificação de vulnerabilidades produzido. Essa ferramenta, não tem como objetivo tornar um sistema seguro; o que queremos é fornecer informações técnicas, de forma sistemática a administradores de *sites Internet*, sobre vulnerabilidades e ataques, indicando como estes podem ser corrigidos e também contribuindo para qualificação profissional dos administradores de *sites Internet*.

A intenção dessa pesquisa não é produzir ferramentas que tornem diretamente um sistema seguro. Procuramos ajudar a reduzir a "ingenuidade" de alguns *sites Internet*, oferecendo uma ferramenta que detecte as vulnerabilidades mais comuns e sugerindo procedimentos para fechá-las. Com isso, será possível contribuirmos para o aumento do nível de segurança de alguns *sites*, além de oferecermos um conhecimento organizado aos administradores desses sistemas sobre os procedimentos básicos de teste para manter mais seguras suas máquinas e informações.

Esse estudo verifica problemas de implementação relacionados à segurança contra acessos não autorizados a sistemas operacionais, enfocando para isso as implementações de algumas aplicações que manipulam os protocolos TCP/IP, montando uma classificação de testes e falhas que possam verificar vulnerabilidades nas implementações acima, assim como nas configurações dos sistemas que as mantêm.

1.2. Trabalhos Relacionados

Administradores de *sites* Internet contam com programas que testam e instalam mecanismos de segurança em sistemas baseados em UNIX. Na próxima seção, descreveremos alguns desses sistemas.

1.2.1. Outras Ferramentas para Verificação de Segurança

a) TRIPWIRE

O *tripwire* é um monitor de integridade de sistemas de arquivos UNIX. Ele usa diversas rotinas para detectar alterações nos arquivos, bem como para monitorar informações mantidas pelo sistema.

A configuração do *tripwire* permite especificar arquivos e diretórios a serem monitorados, ou não, e também permite especificar arquivos onde alterações limitadas são permitidas sem gerar alerta. O *tripwire* também monitora alterações em permissões, *links* e tamanhos de arquivos e diretórios, permitindo, também, detectar adições e deleções de arquivos nos diretórios controlados.

Uma vez instalado em um sistema limpo, ele permite detectar modificações não autorizadas em arquivos, detectando a inserção de *back doors* ou *logic bombs*, bem como a existência de *viruses*. Não requer privilégios de *root* e não modifica o sistema.

b) COPS

O COPS (Computer Oracle and Password System) é um pacote de software que procura por erros de configuração no sistema UNIX. Ele verifica modos/permissões de arquivos, fragilidade de senhas, consistência do /etc/passwd (o COPS verifica, por exemplo, a existência de entradas

duplicadas e contas sem senha), arquivos com SUID de *root setados*, CRC e datas de arquivos binários importantes, entre outros.

c) ISS

O ISS (*Internet Security Scanner*) é um software que testa a vulnerabilidade da rede. Ele vasculha todos os computadores dentro de uma faixa de endereços IP especificada para determinar o nível de segurança oferecido por cada um, levando em conta as vulnerabilidades mais comumente encontradas em um sistema.

O ISS verifica, por exemplo: a existência de contas *default*, que não devem ter senhas triviais; a configuração do *anonymous* FTP, que deve ser feita cuidadosamente, já que, através desse serviço, usuários sem conta em um determinado *site* têm acesso, embora restrito, a certos diretórios do sistema; a configuração do *sendmail* - comandos *sendmail*, como *wiz* e *debug* devem ser desabilitados.

d) SATAN

Outra ferramenta que pode auxiliar no processo de monitoração do estado de segurança do sistema é o SATAN (*Security Analysis Tool for Auditing Network*). Muita informação sobre *hosts* remotos pode ser obitda ao examinar serviços de rede como *finger*, NFS, NIS, FTP e TFTP, e *rexd* (*daemon* para execução remota de comando). Rodando o SATAN numa determinada máquina, todos os *hosts* diretamente conectados a ela podem ser examinados.

Entre os problemas mais encontrados estão:

- sistemas de arquivos (NFS) exportados sem restrições

- antigas versões de *sendmail* (anteriores a 8.6.10)
- *rexd* acessável por qualquer *host*
- diretório *anonymous* FTP aberto para escrita

Para cada tipo de problema encontrado existe um tutorial explicando o que pode ser feito e o seu impacto: corrigir um erro de configuração, instalar um pacote para correção de um *bug*, usar algum meio para restringir acesso ou desabilitar um serviço.

1.3. Organização do Trabalho

O restante deste trabalho foi organizado em 6 capítulos. O capítulo 2 mostra um estudo sobre os aspectos de segurança em redes TCP/IP, enquanto capítulo 3 traz informações sobre os aspectos de segurança em sistemas operacionais Unix, concluindo então a parte que traduz o referencial teórico do trabalho. Seguindo na estrutura do texto, encontramos o Capítulo 4 que mostra um estudo sobre as vulnerabilidades encontradas em *sites Internet* e sua classificação. O Capítulo 5 expõe as características dos principais tipos de ataques a sistemas. No Capítulo 6 apresentamos a especificação do ambiente Groove e suas ferramentas, prototipadas durante o trabalho, que podem ser usadas para testar a segurança de *sites Internet* segundo a classificação criada. Finalmente o Capítulo 7 apresenta algumas considerações finais sobre o trabalho e sugestões para a continuação da pesquisa.

Capítulo 2
Aspectos de Segurança em Redes Baseadas em TCP/IP

2.1. Introdução

O TCP/IP é um conjunto de protocolos desenvolvidos para permitir compartilhamento de recursos entre os computadores de uma rede. O nome mais apropriado para este conjunto de protocolos é suite de protocolos internet.

A arquitetura TCP/IP preconiza a interoperabilidade de redes de forma transparente ao usuário. Isto ocorre devido a algumas características técnicas da arquitetura, tais como:

- Padrões de protocolos abertos, disponíveis gratuitamente e desenvolvidos independentemente de qualquer especificação do *hardware* do computador ou do Sistema Operacional. Por ser largamente difundido, o TCP/IP é ideal para a união de diferentes *hardwares* e *softwares*, até mesmo se estes não estiverem utilizando os recursos da Internet [6].

- Independência da especificação do *hardware* da rede física. Isto permite ao TCP/IP integrar diversos tipos de redes, tais como Ethernet, Token-Ring, FDDI e virtualmente qualquer outro tipo de meio de transmissão física.

- Esquema de endereçamento universal que permite qualquer dispositivo TCP/IP endereçar unicamente qualquer outro dispositivo em toda a rede,

até mesmo em uma rede vasta como a Internet.

- Protocolos de alto nível padronizados para serviços consistentes e amplos disponíveis aos usuários [4].

Essas características possibilitam que computadores com diferentes características de hardware e software se comuniquem entre si. No entanto, ao mesmo tempo que essas características possibilitaram o que hoje nós conhecemos como Internet, o TCP/IP ficou tão popular e suas arquitetura foi tão difundida pelo mundo a fora, que seus problemas também começaram a ser observados. Dentre esses problemas estão os que comprometem a segurança de determinado sistema. A partir de agora, faremos uma rápida referência à arquitetura TCP/IP e seu funcionamento para logo em seguida, ainda neste capítulo, apontarmos alguns dos problemas de segurança causados por implementações TCP/IP.

2.2. Arquitetura TCP/IP

Não existe um acordo universal sobre como descrever a Arquitetura TCP/IP como um modelo de camadas; descrições do TCP/IP geralmente definem de três a cinco níveis funcionais na arquitetura. O modelo de quatro camadas ilustrado na figura 2.1 provê uma representação da hierarquia de camadas dos protocolos TCP/IP [6].

Camada de Aplicação	Consiste das aplicações e dos processos que utilizam a rede
Camada de Transporte	Provê serviços de entrega de dados do emissor ao receptor
Camada de Rede	Define os *datagramas* e realiza o roteamento dos dados
Camada Física	Define os métodos para acessar a rede física

Figura 2.1 - As camadas ou níveis do TCP/IP

Esta estrutura do TCP/IP é vista do modo como os dados são manuseados. Os dados descem a pilha dos protocolos desde a camada de aplicação até a camada física. Cada camada da pilha adiciona informação de controle para garantir uma transmissão correta. Esta informação de controle é chamada **Cabeçalho** (*Header*), quando está posicionada na frente dos dados a serem transmitidos e **Finalizador** (*Trailer*) quando aparece no final dos dados. Cada camada trata as informações recebidas da camada acima como dados e adiciona seu próprio cabeçalho na frente da informação. Esta adição da informação transportada em cada camada é chamada **Encapsulamento**, como nos ilustra a figura 2.2.

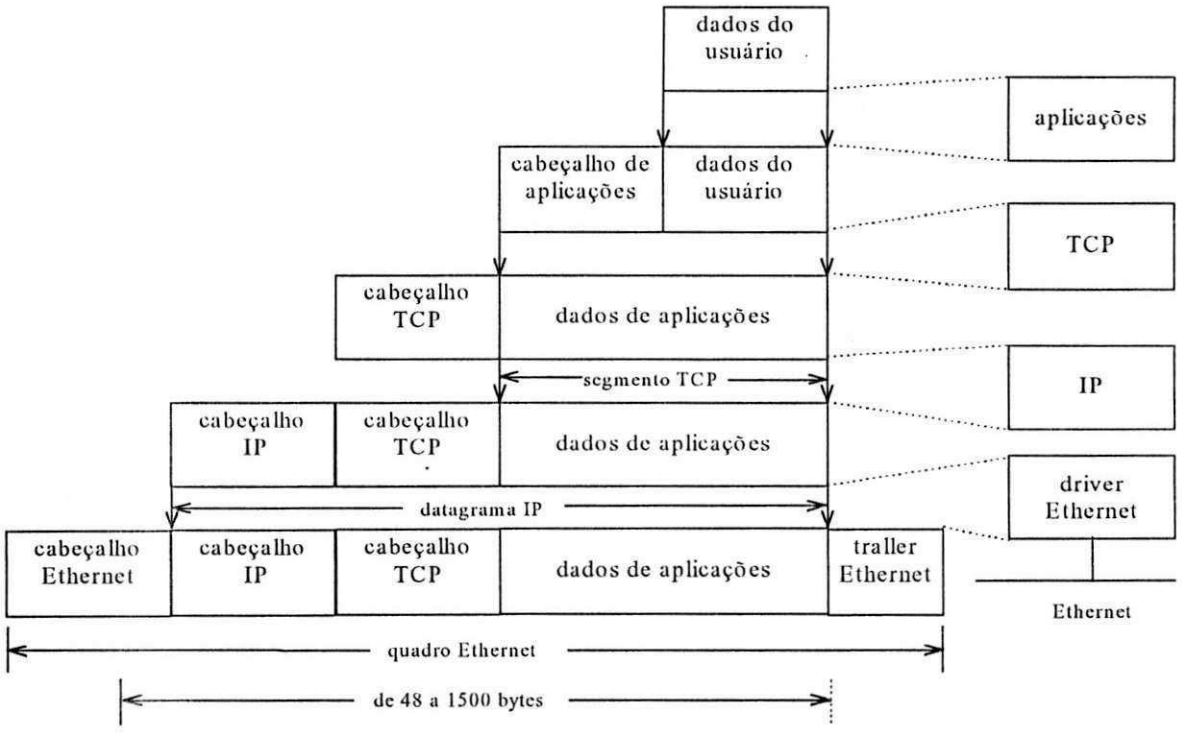


Figura 2.2: Exemplo de encapsulamento em uma rede TCP/IP sobre ethernet

2.2.1 Camadas da *suite* Internet

As camadas da arquitetura Internet são apresentadas na figura 2.3.

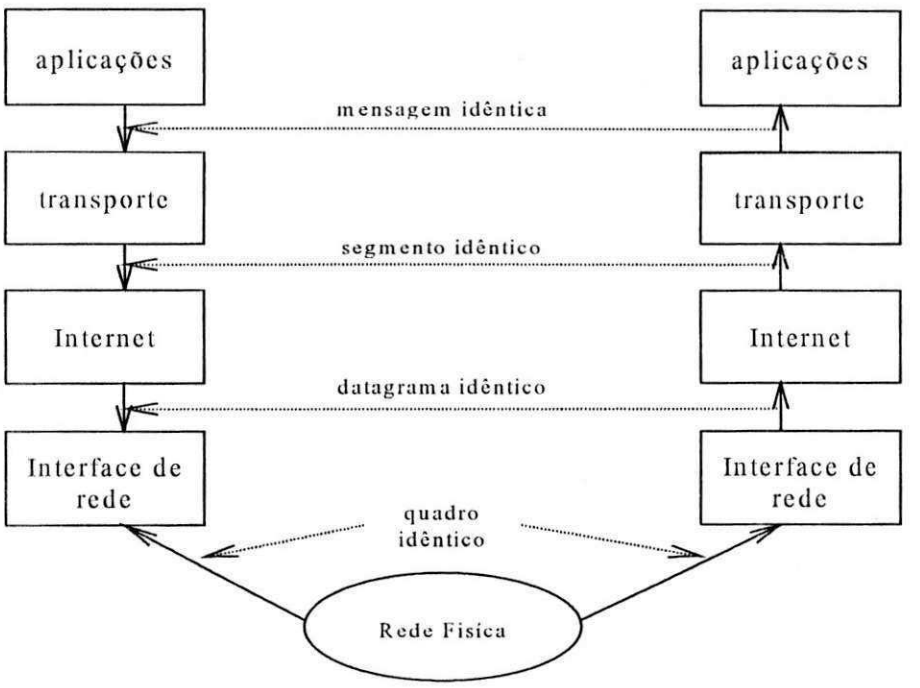


Figura 2.3 - Arquitetura Internet

Verificamos que uma camada pode se comunicar com diferentes camadas vizinhas, isto é, com distintos protocolos das camadas superior e inferior simultaneamente.

2.2.1.1. Camada ou Nível Físico

Esta é a camada mais inferior na hierarquia da arquitetura TCP/IP. Os protocolos desta camada permitem o acesso ao sistema para enviar os dados aos outros dispositivos conectados diretamente a uma rede. Esta camada define como usar a rede para transmitir um *Datagrama* IP. Diferente dos protocolos de alto nível, os protocolos desta camada devem conhecer os detalhes da rede subjacente (estrutura de pacotes, endereçamento, etc.) para formatar corretamente os dados que serão transmitidos, com a finalidade de evitar incompatibilidade com as restrições da rede.

2.2.1.2. Camada ou Nível de Rede

A camada acima da camada física na hierarquia da Arquitetura TCP/IP é a camada de rede. O protocolo Internet (IP) é o mais importante protocolo deste nível. O IP fornece o pacote básico de serviço de transmissão no qual as redes TCP/IP são construídas.

O IP é um protocolo sem conexão. Isto significa que o IP não troca informações de controle (*handshake*) para estabelecer uma conexão fim-a-fim antes de transmitir os dados.

O IP confia em outros protocolos para oferecem detecção e recuperação de erros, por isto o IP é chamado também de um protocolo não confiável, pois ele não checa se os dados são recebidos corretamente.

Quando tratamos de endereçamento no nível de rede, estamos trabalhando em um nível virtual. Portanto, o endereçamento utilizado é configurado e parametrizado via *software*.

O IP move dados entre máquinas na forma de *datagramas*, onde cada um desses é transmitido para o endereço contido no campo Endereço de Destino (*Destination Address*) do *datagrama*, um campo de 32 bits.

2.2.1.3. Camada ou Nível de Transporte

O principal objetivo da camada de transporte é oferecer a transferência confiável de dados entre diferentes computadores pertencentes ou não a mesma rede, garantindo que os dados sejam entregues sem erros, sem perdas ou duplicação.

Nesta camada são utilizados dois protocolos. O protocolo orientado a conexão - **TCP** (Protocolo de Controle de Transmissão), e o protocolo não orientado a conexão - **UDP** (Protocolo de *Datagrama* de Usuário), que são

especificados respectivamente nas **RFCs**[1] **761** e **768**. A utilização destes protocolos varia de acordo com a necessidade da aplicação considerada. Por exemplo, aplicações de gerenciamento utilizam UDP, e outras, como transferência de arquivos, utilizam TCP [5].

Estabelecimento da Conexão de Transporte

Ao estabelecer uma conexão, o TCP utiliza o mecanismo *Three Way Handshake* (Aperto de Mão Triplo) que pode ser iniciado por qualquer uma das partes envolvidas [4].

Este mecanismo é necessário e suficiente para garantir uma sincronização correta entre as duas pontas da conexão. A justificativa para esta afirmação é que o TCP opera sobre o serviço IP, que não é confiável podendo perder, duplicar, retardar ou entregar os dados fora da ordem original. Deste modo o TCP deve usar um mecanismo para temporizar e retransmitir os dados perdidos. Estes problemas acontecem enquanto uma conexão está sendo estabelecida, usada ou encerrada.

O funcionamento do *Three Way Handshake* pode ser visualizado na figura 2.4.

[1] RFC: Request for Coment - Documentos que especificam o funcionamento dos protocolos e serviços TCP/IP.

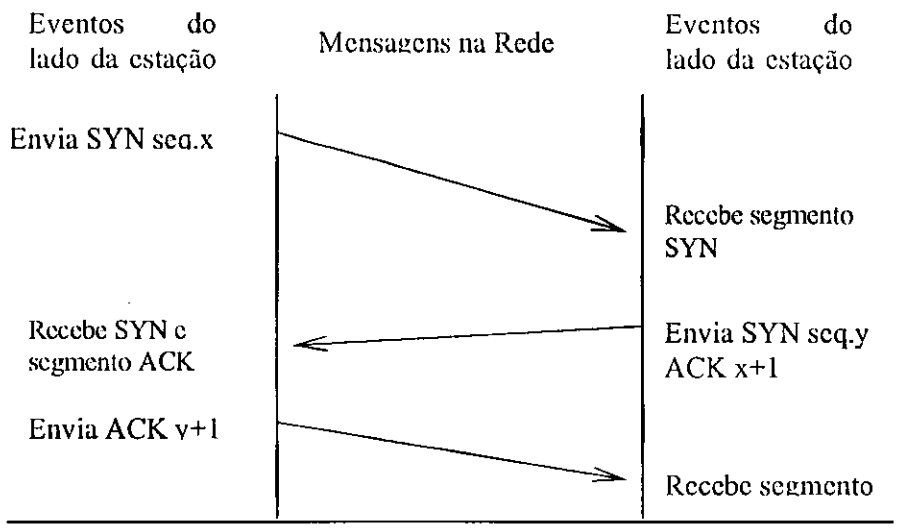


Figura 2.4 - Funcionamento do Three Way Handshake

Pode-se resolver os problemas citados utilizando o *Three Way Handshake*, que utiliza números de sequência no estabelecimento de uma conexão, pois não podem ser estabelecidas conexões seguidas com os mesmos números de sequência, portanto isto evita que os segmentos atrasados sejam interpretados como vindos de uma nova conexão.

O campo *code bit* do cabeçlho TCP, onde o *bit SYN* representa uma solicitação de transmissão, identifica o primeiro segmento da conexão. Se o *bit SYN* estiver selecionado em um, deve haver um segmento ACK que contém o número do campo *sequence number* do segmento recebido acrescido de 1 (um). Para realizar o fechamento da conexão o emissor do segmento SYN transmite um ACK para o receptor.

Por exemplo, na figura 2.4, a estação 1 envia uma requisição **SYN** (requisição de sincronismo) mais um número sequencial (sequence number) **seq.x** para a estação 2; ao receber esta requisição, a estação 2 envia para a 1 uma resposta contendo uma requisição **SYN**, seu número sequencial **seq.y** e um **ACK** (uma forma de informar a estação 1 que 2 recebeu sua requisição). Este **ACK** é representado por um número gerado pelo acréscimo de 1 unidade ao número sequencial emitido pela estação 1. Após a estação 1 receber esta mensagem da estação 2, 1 responde também com um ACK

formado pela soma de uma unidade ao número sequencial emitido por 2, estabelecendo assim, a conexão TCP.

Em uma conexão estabelecida os dados fluem no modo *full-duplex*, e os envolvidos concordam com os números de sequência. A partir do número inicial, que é escolhido aleatoriamente todo o fluxo de dados o tomará como referência inicial.

Na arquitetura TCP/IP, o protocolo UDP oferece mecanismos primários que os programas de aplicação utilizam para enviar *datagramas* para outra aplicação. Sendo assim, transporta uma mensagem de uma estação para outra, utilizando o IP, situado na camada inferior, e oferece o mesmo nível de confiabilidade do protocolo IP.

2.2.1.4. Camada de Aplicação

Dentre as funções podemos citar a transferência de arquivos, onde a camada desempenha o papel de compatibilizar a transferência entre sistemas diferentes, autenticar os parceiros de comunicação, onde verifica-se se o usuário é ou não autorizado a utilizar os serviços prestados e acessar os recursos associados desta camada.

Os protocolos de aplicação, que são implementados no topo da arquitetura TCP/IP, são responsáveis por abrir a conexão para um computador específico, conectar-se a ele, informando o que deseja, e controlar a transmissão do arquivo. Isto é, quando deseja-se enviar uma mensagem, os protocolos de aplicação as enviam para o TCP, o qual garantirá o envio para o destino correto [10].

2.3. Problemas com redes TCP/IP

Alguns ataques podem ser realizados contra qualquer rede que se conecte à Internet sem a devida proteção. A maioria deles é passível de ser realizada

facilmente por qualquer pessoa que disponha do conhecimento técnico necessário.

Alguns firewalls protegem sua rede contra todos os ataques aqui mencionados e muitos outros, oferecendo uma total transparência e flexibilidade. No entanto, o nível de proteção oferecido por um firewall é diretamente proporcional ao nível de limitações que são impostas aos usuários do sistema.

2.2.1.Varreduras de portas

A primeira coisa que um atacante normalmente faz em uma rede é o chamado Varredura de Portas. Isto consiste em enviar pacotes para todas as portas de uma máquina (ou de várias máquinas de uma rede), de modo a descobrir quais são os serviços oferecidos por cada uma delas. Com o uso desta técnica, é possível se determinar exatamente quais serviços TCP e UDP estão sendo oferecidos.

Este tipo de ataque ficou muito popular quando em 1995 foi lançado, com grande publicidade, o software SATAN, desenvolvido por Dan Farmer e Wietse Wenema. Feito em Perl, C e HTML, com uma interface amigável, o SATAN faz uma varredura completa em uma máquina ou rede denominada "alvo", relatando todos os serviços oferecidos e pontos vulneráveis. Ele não só descobre quais serviços estão disponíveis como também utiliza estes serviços para descobrir mais informações sobre o alvo e mais pontos vulneráveis. Existem atualmente inúmeros programas com funcionalidade similar ao SATAN, alguns em domínio público.

2.2.2. IP Spoofing

IP Spoofing é o nome dado para falsificações de endereços IP, fazendo com que um pacote seja emitido com o endereço de origem de uma outra

máquina. Com o uso desta técnica, é possível para um atacante assumir a identidade de qualquer máquina da Internet.

O perigo maior deste ataque é para serviços baseados no protocolo UDP. Muitos serviços baseados neste protocolo só permitem acesso para determinadas máquinas, explicitamente configuradas pelo administrador. Este é o caso do serviço NFS, que permite o compartilhamento de discos remotos. (vamos dar um exemplo de ataque usando este serviço específico, mas o ataque é válido para qualquer serviço similar)

Um atacante pode enviar um pacote UDP para a máquina servidora de NFS como se este viesse de uma máquina cliente, autorizada a gravar em algum diretório. Este pacote pode conter solicitação de criação, remoção e escrita de arquivos. Como a máquina servidora acha que o pacote vem da máquina cliente (por causa do IP de origem falsificado), todas as solicitações serão prontamente atendidas. Este tipo de ataque não pode ser bloqueado sem a existência de um firewall.

2.2.3. Roteamento dirigido

O protocolo TCP/IP foi um protocolo criado na época da guerra fria, com o objetivo de conectar a rede ARPANET, a rede de defesa do exército americano. Um dos objetivos desta rede, e do protocolo TCP/IP, era permitir que a mesma continuasse funcionando, mesmo que uma parte dela fosse derrubada (isto era fundamental no caso de ataques). Para possibilitar este tipo de comportamento, o protocolo IP foi dotado da capacidade do emissor de um pacote estabelecer a rota que o mesmo deveria seguir ao ser enviado através da rede. Desta forma, mesmo que uma parte da rede fosse derrubada, poderíamos estabelecer caminhos alternativos para o tráfego e manter a comunicação.

A quase totalidade das implementações de TCP/IP atuais ainda seguem as especificações originais, que determinam que ao se receber um pacote com

roteamento dirigido, todas as repostas à este pacote também devem ser direcionadas e devem seguir a mesma rota especificada no pacote de origem (na ordem inversa, evidentemente). Se isso não fosse desta forma, não seria possível se atingir o objetivo de manter a comunicação em funcionamento.

Apesar de interessante, este comportamento possibilita a existência de um tipo de ataque que é fácil de ser implementado e impossível de ser impedido, senão com o uso de algum tipo de firewall. O ataque se baseia no fato de que em muitas vezes a validação de um serviço ou de um usuário é feito com base no endereço IP da máquina que está se conectando. Com o uso de pacotes direcionados, um atacante pode enviar pacotes para as máquinas da rede interna como se fossem originados de uma máquina confiável. Desta forma, o atacante consegue estabelecer uma conexão válida para uma máquina da rede interna com os direitos que teria se estivesse se conectando a partir de uma outra máquina.

Este tipo de ataque é particularmente perigoso em serviços que utilizam apenas endereços IP para fazer a validação de usuários, como o rsh, rlogin, etc.

2.2.4. Ping O'Death

Este ataque foi descoberto recentemente e bastante explorado na Internet. Ele consiste em se enviar um pacote IP com tamanho maior que o máximo permitido (65535 bytes), para a máquina que se deseja atacar. O pacote é enviado na forma de fragmentos (a razão é que nenhum tipo de rede permite o tráfego de pacotes deste tamanho) e quando a máquina destino tenta montar estes fragmentos, inúmeras situações podem ocorrer: a maioria da máquinas trava, algumas reinicializam, outras abortam e mostram mensagens no console, etc. Praticamente todas as plataformas eram afetadas por este ataque, e todas as que não tiveram correções de segurança instalados, ainda o são.

Este ataque recebeu o nome de Ping O' Death porque as primeiras ocorrências deste ataque foram a partir do programa ping, entretanto, qualquer pacote IP com mais de 65535 (pacote inválido) provoca o mesmo efeito.

Alguns Firewalls impedem este ataque na medida em que eles armazenam e montam todos os fragmentos de pacotes IP recebidos. Quando eles detectam um pacote inválido, o mesmo é automaticamente descartado.

2.2.5. SYN Flood

SYN Flood é um dos mais populares ataques de negação de serviço (denial of service). Esses ataques visam impedir o funcionamento de uma máquina ou de um serviço específico. No caso do SYN Flood, consegue-se inutilizar quaisquer serviços baseados no protocolo TCP.

Para se entender este ataque, é necessário primeiro se entender o funcionamento do protocolo TCP, no que diz respeito ao estabelecimento de conexões, detalhado anteriormente.

Todos os pedidos de estabelecimento de conexões recebidos por um servidor ficam armazenados em uma fila especial, que tem um tamanho pré-determinado e dependente do sistema operacional, até que o servidor receba a comunicação da máquina cliente de que a conexão está estabelecida. Caso o servidor receba uma pedido de conexão e a fila de conexões em andamento estiver cheia, este novo pacote de pedido de abertura de conexão é descartado.

O ataque consiste basicamente em se enviar um grande número de pacotes de abertura de conexão, com um endereço de origem forjado, para um determinado servidor. Este endereço de origem é forjado para o de uma máquina inexistente (muitas vezes se usa um dos endereços reservados descritos no capítulo sobre conversão de endereços). O servidor, ao receber

estes pacotes, coloca uma entrada na fila de conexões em andamento, envia um pacote de resposta e fica aguardando uma confirmação da máquina cliente. Como o endereço de origem dos pacotes é falso, esta confirmação nunca chega ao servidor.

O que acontece é que em um determinado momento, a fila de conexões em andamento do servidor fica lotada. A partir daí, todos os pedidos de abertura de conexão são descartados e o serviço inutilizado. Esta inutilização persiste durante alguns segundos, pois o servidor ao descobrir que a confirmação está demorando demais, remove a conexão em andamento da lista. Entretanto, se o atacante persistir em mandar pacotes seguidamente, o serviço ficará inutilizado enquanto ele assim o fizer.

Alguns firewalls possuem um módulo especial de proteção contra ataques de SYN flood, oferecendo total proteção contra ataques deste tipo.

2.2.6. Ataques contra o protocolo NETBIOS

Praticamente todas as redes locais existentes hoje em dia possuem PCs rodando plataformas Windows 3.11(tm), Windows 95(tm), Windows NT(tm) ou OS/2(tm). Estas plataformas possuem a característica de disponibilizar todos os seus serviços de rede através do protocolo NETBIOS. Ocorre também que praticamente todas estas máquinas rodam o protocolo TCP/IP, possibilitando que o NETBIOS seja encapsulado sobre o TCP/IP. Se estas máquinas estão conectadas à Internet, é possível para um atacante verificar quais são os diretórios e impressoras compartilhadas por cada uma delas.

Além disso, devido ao fato de muitas vezes os compartilhamentos serem feitos indiscriminadamente, normalmente pelos próprios usuários, é muito comum se conseguir acesso de leitura ou de gravação em muitos diretórios. Desta forma um atacante pode ter acesso à informações confidenciais, apagar informações importantes ou mesmo implantar arquivos contaminados nas máquinas da rede interna.

Até pouco tempo atrás, as máquinas Windows NT eram vulneráveis a diversos tipos de ataques realizados contra os serviços NETBIOS, que terminavam por travar o sistema ou inutilizar completamente todos os serviços (os ataques podiam ir de um simples telnet para a porta 139 até a ataques mais sofisticados).

Mesmo hoje em dia ainda existem muitas máquinas vulneráveis, devido ao fato de ser necessário a aplicação de correções específicas para estes problemas, coisa que muitos administradores não fazem.

A maioria dos firewalls modernos bloqueiam ataques deste tipo facilmente: os serviços NETBIOS utilizam as portas 137, 138 e 139 dos protocolos TCP e UDP e portanto basta bloquear o acesso destas portas para máquinas externas.

2.2.7. Ataques contra o X-Windows

X-Windows é o padrão de interface gráfica utilizado em máquinas Unix. Ele é um padrão que foi criado visando o seu uso em redes locais. Desta forma, é possível se executar um programa em uma máquina e mostrar sua saída em outra máquina, localizada em qualquer lugar da Internet. É também possível um usuário ter programas sendo executados em diversas máquinas e todos eles sendo visualizados como se fossem aplicações locais, cada um na sua janela. Toda esta flexibilidade, entretanto, tem problemas de segurança: caso disponha de autorização do servidor X-Windows (a máquina onde está a tela gráfica), é possível para um atacante exercer um controle completo sobre o tela, capturando imagens e caracteres ou mesmo enviando caracteres para outras aplicações. É possível até mesmo simular o movimento do mouse ou encerrar outras aplicações.

Esta autorização muitas vezes não é difícil de ser conseguida: o X-Windows possui dois métodos de autenticação, um a nível de usuário e outro a nível de máquina. Quando se está usando autenticação a nível de máquina, um

Um ataque típico seria repassar, para os roteadores da rede, informação de que os pacotes enviados para uma determinada rede devem ser roteados através dele, atacante, que posteriormente os envia para a destinação correta. Durante a passagem dos pacotes, o atacante é capaz de examinar seu conteúdo em busca de senhas ou informações interessantes ou mesmo alterar seu conteúdo.

Um firewall é capaz de bloquear ataques deste tipo, considerando que os pacotes RIP são pacotes UDP enviados para porta 513 e, portanto, basta bloquear pacotes para esta porta para que o ataque não possa ser efetivado.

2.2.9. Ataque de Fragmentação

Um ataque de fragmentação na verdade não é um ataques propriamente dito. Ele é uma maneira de fazer com que serviços TCP de máquinas protegidas por um filtro de pacotes possam ser acessados mesmo que o filtro não permita seu acesso (ou seja, ele é uma maneira de enganar o filtro de pacotes).

Boa parte dos filtros de pacotes tradicionais apenas filtra o primeiro fragmento de um pacote, uma vez que sem este fragmento não será possível para a máquina cliente montar o pacote completo, e deixa os demais fragmentos passarem. Além disso, se o primeiro fragmento não for um pacote de abertura de conexão, ele também é autorizado a passar.

O ataque consiste em se fragmentar o pacote TCP de modo que o segundo fragmento contenha parte do cabeçalho TCP, mais especificamente as portas e os flags do protocolo. Este segundo fragmento possuirá os flags para uma abertura de conexão. O primeiro fragmento consiste em um pacote que não é de abertura de conexão.

Da forma com que ambos os fragmentos foram montados, eles serão aceitos pelo filtro de pacotes e atingirão a máquina destino. Uma vez na máquina

destino, os pacotes serão montados, produzindo um pacote de abertura de conexão, que se estivesse sido enviado inteiro teria sido bloqueado no filtro de pacotes.

2.2.10. Varredura invisível

A varredura invisível (stealth scanning) é uma maneira de um atacante descobrir todos os serviços TCP que estão rodando em uma rede, mesmo que esta esteja sendo protegida por um filtro de pacotes (não será possível ser conectar aos serviços utilizando esta técnica, mas ela já um primeiro passo para um ataque).

O ataque consiste em fazer uma varredura de portas (como já mostrado), porém com a diferença de não se enviar pacotes de abertura de conexão e sim pacotes simulando uma conexão existente. Como os filtros de pacotes normalmente permitem o trânsito dos pacotes que não são abertura de conexão, todos estes pacotes atingiriam a máquina destino. A máquina destino, ao receber os pacotes, reagiria de maneira diferenciada caso existisse um serviço rodando na porta especificada ou não.

Um firewall pode impedir este tipo de ataque desde que mantenha um registro de todas as conexões abertas e só permite a passagem de pacotes para estas conexões.

2.2.11. Transparência dos dados na rede

A Internet é uma rede mundial composta de milhares de máquinas espalhadas por todo o mundo. Quando duas máquinas quaisquer estão se comunicando, todo o tráfego entre elas passa de máquina em máquina da origem até o destino. Na quase totalidade das vezes, a administração destas máquinas intermediárias é feita por terceiros e nada se pode afirmar quanto a sua honestidade. (na maioria das vezes, não é nem possível se saber de antemão por quais máquinas os pacotes passarão até atingir o destino).

O problema que isso causa, é que todo o tráfego entre as duas máquinas que estão se comunicando pode ser visualizado ou alterado por qualquer uma das máquinas intermediárias. Este problema se torna ainda mais crítico quando se pretende transmitir dados confidenciais ou críticos para a operação das duas entidades se comunicando.

Esse problema ainda não conseguiu ser totalmente resolvido por soluções existentes hoje no mercado, considerando que essa solução resume-se a tornar um canal de comunicação de dados seguro usando chaves criptográficas e, como sabemos, existe o velho problema recursivo de distribuição de chaves.

Capítulo 3
Aspectos de Segurança em Sistemas Operacionais UNIX

3.1. Introdução

Em sistemas UNIX, os mecanismos de segurança herdaram características de sistemas multiusuário e multitarefa, protegendo arquivos e diretórios de usuários através de senhas de acesso e permissões de arquivos.

Essas características de segurança dos sistemas UNIX acabam fazendo com que todo seu esquema de segurança fique delegado a usuários e administradores, criando uma dependência direta entre o estado de segurança e a qualificação do usuário e do administrador [15].

Neste capítulo, descreveremos potenciais problemas causados por características de segurança de sistemas UNIX.

3.2. Ameaças Programadas

Computadores são projetados para executar instruções normalmente associadas com atividades como cálculo de valores, manutenção de bases de dados, comunicação com usuários e com outros sistemas, etc. Algumas vezes, contudo, as instruções executadas podem ser danosas ao sistema. Quando o dano acontece por acidente, nós dizemos que ocorreu uma falha de *software*. Porém, se a fonte da instrução danosa é um indivíduo que

pretende que algo de anormal aconteça no sistema, nós chamamos essa instrução de ameaça programada [12].

Existem diferentes tipos de ameaças programadas. Especialistas classificam ameaças pela forma como elas se comportam. Recentemente, ocorrências dessas ameaças programadas têm sido descritas pela mídia como vírus. Contudo, vírus fazem somente uma parte das operações atribuídas a uma ameaça programada.

Os principais tipos de *ameaças* programados são:

- **Back door (porta dos fundos)**, falhas de implementação que permitem acesso não autorizado a sistemas.

- **Logic bombs (bombas lógicas)**, que são funcionalidades embutidas em programas e acionados por certas condições.

- **Vírus**, ou programas que modificam outros programas, inserindo cópias deles mesmos.

- **Worms (vermes)**, programas que se propagam de computador a computador numa rede, sem necessariamente modificar outros programas nas máquinas alvo.

- **Trojan horses (cavalos de tróia)**, ou programas que parecem ter uma função, mas realmente têm outra.

Algumas das ameaças mencionadas acima também podem possuir comportamento não destrutivo. Por exemplo, *worms* podem ser usados para fazer computação distribuída em processadores ociosos, *back doors* são muito usadas para depurar programas e vírus podem ser escritos para atualizar código fonte. Portanto, não é a abordagem, mas sim o propósito de uma ameaça programada que pode torná-la nociva [14].

Nesse capítulo descreveremos cada uma das ameaças mencionadas acima, mostrando como elas afetam um sistema UNIX, e expondo como podemos nos proteger delas.

3.2.1. Back Doors (Portas dos Fundos)

Back Doors[1], também chamados de *trap doors*, são partes de código escritos em aplicações ou sistemas operacionais para garantir acesso de programadores a programas sem usar os métodos de autenticação de acesso normais. *Back doors* são usadas há muito tempo. Elas são escritas tipicamente por programadores de aplicação que necessitam de uma forma de depurar ou monitorar o código que eles desenvolvem [17].

A maioria das *back doors* são inseridas em aplicações que requerem grandes procedimentos de autenticação, ou longos *setups*, requerendo que um usuário forneça alguns valores diferentes para executar a aplicação. Quando depurando um programa, o desenvolvedor pode escolher por obter privilégios especiais que possam ignorar os procedimentos de autenticação ou então por seguir todos os passos desse procedimento. O programador pode também querer que exista uma forma de ativar o programa sem ativar o procedimento de autenticação, mesmo sem privilégios especiais. Uma *back door* é o código que permite o reconhecimento de uma seqüência especial de entrada, garantindo assim, um acesso especial.

Back doors passam a ser ameaças quando elas são usadas por programadores inescrupulosos para obter acessos não autorizados, ou quando o programador da aplicação esquece de remover uma *back door*

[1] Back doors: Mecanismos de programação utilizados por desenvolvedores durante a produção de código fonte de aplicações, visando testar seus programas. Uma back Door pode ser esquecida por um programador, gerando uma vulnerabilidade no sistema que pode ser explorada por um atacante com o conhecimento da sequência de comandos usada pelo programador durante a fase de testes.

após terminar a depuração e algum indivíduo descobre a existência de tal mecanismo.

A mais famosa *back door* do UNIX foi a opção DEBUG do programa *sendmail*, explorada por um programa *worm* em novembro de 1988. A opção *"debug"* foi adicionada para depurar o *sendmail*. Infelizmente, a opção *"debug"* tinha uma *back door* nela, a qual permitia acesso remoto de computadores a uma rede, sem necessitar do procedimento de *login* [13].

A opção *debug* foi acidentalmente habilitada em versões do sendmail distribuídas pela *Sun Microsystem*, Digital Equipament Corporation e outras.

Algumas vezes, um invasor insere uma *back door* em um sistema depois que ele consegue o acesso ao sistema. A *back door* possibilita ao invasor voltar ao sistema.

Back doors se apresentam de algumas formas:

- ✓ Instalar e alterar versões de *login, telnetd, ftpd, rshd* ou algum outro programa. O programa alterado normalmente aceita uma seqüência especial de entrada e fornece um *shell* para o usuário [12].

- ✓ Criar uma entrada no arquivo *.rhosts*[2] de um usuário ou do superusuário para permitir futuros acessos não autorizados ao sistema alvo.

- ✓ Mudar o arquivo */etc/fstab* em um sistema NFS (Sistema de Arquivos de Rede) para remover o designador NOSUID, permitindo a um usuário do sistema ganhar direitos de *root* sem autorização.

[2] ..rhosts: arquivo do usuário, contendo uma lista com os nomes dos possíveis computadores remotos de onde o usuário pode se conectar sem necessitar do procedimento de autenticação.

✓ Adicionar um *alias* ao sistema de *mail*, de tal forma que quando um *mail* for transmitido para aquele *alias*, o programa *sendmail* execute um programa projetado pelo atacante que possivelmente cria uma entrada no sistema.

✓ Mudar o proprietário do diretório */etc*, de forma que o atacante possa remover e/ou corromper arquivos como o */etc/passwd* e */etc/group*.

✓ Mudar ou adicionar um serviço de rede para obter um *shell* de *root* através de uma chamada remota.

De uma forma geral, um atacante pode alterar arquivos e configurações de um sistema usando *back doors*, de tal forma que o administrador não tenha como detectar as alterações feitas.

Proteger sistemas UNIX de *back doors* é complicado. Devemos checar sempre:

- A integridade de arquivos importantes.
- Procurar periodicamente por arquivos com SUID/SGID[3].
- Verificar permissões de arquivos importantes.

3.2.2. Bombas Lógicas

Bombas Lógicas são ameaças programadas que possuem sua funcionalidade acionada por parâmetros bem definidos, por isso podem ficar "escondidas" durante muito tempo [16].

[3] SUID: Super User Identification. Um arquivo com SUID ligado pode ser executado por um determinado usuário e, executar sua tarefa com os privilégios de um outro usuário especificado em seu UID.
SGID: Super User Group Identification.Um arquivo com o SGID ligado pode ser executado por um determinado usuário e, executar suas tarefas com os privilégios do grupo especificado em seu GID

Condições geralmente usadas para acionar funcionalmente uma bomba lógica incluem a presença ou ausência de certos arquivos, um dia da semana em particular ou um determinado usuário executando uma aplicação. Uma bomba lógica verifica primeiro quais os usuários logados ou quais os programas que estão em uso no sistema, podendo então destruir ou alterar dados, paralizar ou ainda danificar o sistema operacional [21].

Temporizadores são muito usados por bombas lógicas e são ocasionalmente usados para forçar pagamentos ou outras posições contratuais. Temporizadores fazem programas pararem de executar após um certo tempo, além de executarem ações especiais [20].

Para proteger sistemas de bombas, não devemos instalar *software* sem licença ou sem testá-lo e devemos fazer cópias regulares de arquivos principais do sistema visando facilitar a recuperação das informações no caso de alguma alteração causada por esse tipo de ameaça programada [19].

3.2.3. Vírus

Vírus são sequências do código que são inseridos em determinados programas, de tal forma que, quando esses programas estão executando, o código do vírus é também executado. O código do vírus faz com que uma cópia dele mesmo seja inserido em um ou mais programas. Vírus não são programas distintos, eles não podem executar sozinhos, necessitam que algum outro programa, do qual eles fazem parte, esteja em execução.

Vírus são fenômenos relativamente comuns e são mais encontrados em computadores pessoais executando sistemas operacionais não protegidos, tais como o MS-DOS e o Windows95. Vírus para ambientes UNIX são mais difíceis de serem encontrados, no entanto, existem registros de vírus escritos para UNIX, mesmo que apenas como curiosidade intelectual, considerando que as experiências não relatam danos aos sistemas [25].

Para protegermos sistemas contra ataques de vírus, podemos usar algumas das técnicas usadas para combater *back doors*:

✓ Executar programas de verificação de integridade no sistema.

✓ Não incluir diretórios e programas em *paths* de execução aleatoriamente.

✓ Não deixar os diretórios *bin* (*/bin, /usr/bin, /usr/ucb*, /etc) desprotegidos, ou seja, com permissões de escrita habilitadas.

✓ Selecionar as permissões de arquivos de comandos para modos que permitam que um usuário comum possa apenas lê-los e executá-los, visando protegê-los de alterações não autorizadas.

✓ Não carregar código binário na máquina, a menos que a fonte seja confiável.

✓ Fazer com que somente o usuário tenha direitos de escrita em seu diretório.

3.2.4 . Vermes (*worms*)

Vermes são programas que podem executar de forma independente e que transitam de máquina a máquina dentro de uma rede. Vermes podem ter partes deles mesmos executando em máquinas diferentes. Vermes não mudam outros programas, eles podem transportar código entre máquinas (por exemplo, podem transportar um vírus). Vermes são difíceis de escrever, porém podem causar grandes danos. Desenvolver um programa desse tipo requer um ambiente de rede e um autor que esteja familiarizado não apenas com serviços e facilidades de rede, mas também com facilidades operacionais requeridas para suportá-lo nas máquinas envolvidas [21].

A proteção contra programas verme é parecida com a proteção contra *break-ins* (paradas de sistema). Se um invasor pode entrar na máquina, então um programa verme também pode. Se a máquina está segura contra acessos não autorizados, ela deverá estar segura contra programas verme [17].

Todas as técnicas usadas para evitar acesso remoto à máquina são também aplicadas aqui para proteger o sistema contra vermes.

3.2.6. Cavalos de Tróia (Trojan Horses)

Um programa Cavalo de Tróia pode ser qualquer programa que um usuário venha a utilizar – um jogo, um editor de texto, etc. Enquanto o programa aparentemente executa aquilo que o usuário pediu, ele também está executando algo que o usuário não sabe – que pode ser danoso ao sistema [21].

Por exemplo, o usuário pode pensar que o programa é um jogo. Enquanto ele responde às questões "Quantos jogadores vão jogar?" e "Qual o nível de dificuldade do jogo?", o programa pode estar removendo arquivos, reformatando o disco ou ainda alterando informações no sistema.

Um exemplo foi enviado como um arquivo com formato de *script shell* para um grupo de estudo de código fonte UNIX há poucos anos atrás. O arquivo era grande e continha comandos para descompactar um número de arquivos em um diretório local. Contudo, existiam linhas nesse arquivo, que continham sequências de comandos tipo [28]:

```
rm –rf $HOME
echo Boom!
```

Alguns *sites* relataram problemas, pois executaram a descompactação de tal arquivo como usuário *root*. Como o comando rm pertencente aos sistemas UNIX tem a finalidade de apagar determinados arquivos, o *script* acima é

responsável pela remoção recursiva de todos os arquivos e diretórios indicados pela variável de ambiente HOME, que no caso de alguns sistemas UNIX, para o usuário *root* possui o valor */*. Isso indica que o resultado final da execução desse *script* é a remoção recursiva de todos os arquivos e diretórios encontrados abaixo do diretório raiz (*/*) do sistema.

Devemos sempre lembrar que é possível colocar comandos em outros lugares que não sejam nos diretórios onde ficam os programas compilados. Arquivos *shell* (especialmente arquivos *shar*), *awk, perl* e *scripts sed.* Arquivos *Tex*, arquivos *postscript* e alguns *buffers* de editores também podem conter comandos danosos.

Comandos colocados em *buffers* de editores apresentam um problema potencial. Alguns editores permitem que códigos de controle sejam colocados nas primeiras ou nas últimas linhas de arquivos para inicializar automaticamente o editor e executar comandos. Colocando as linhas de comando apropriadas, podemos causar danos a um sistema toda vez que um dado usuário ler o *buffer* de seu editor [28].

Uma outra forma de Cavalo de Tróia é usar "comandos de blocos transmitidos" ou o "modo de resposta pra trás" em terminais. Alguns tipos de terminais suportam modos onde certas sequências de caracteres de controle fazem com que a linha corrente ou a linha de status sejam executadas como se tivessem sido digitadas pelo teclado. Então, o seguinte comando pode ser colocado em um mail e enviado:

rm –rf $HOME & logout <clear screen, send sequence>

Quando a vítima ler seu *mail*, a linha é ecoada para trás como um comando para ser executado no próximo prompt. Ao logar novamente, a vítima constata o dano. Se seu terminal permite essa funcionalidade, aconselhamos desabilitá-la.

Para proteger sistemas de Cavalos de Tróia, é necessário que nunca façamos ações de edição, descompactação ou execução de programas externos, com permissões do usuário *root* e que procuremos entender exatamente o que um arquivo faz para depois executá-lo [24].

3.3. Funcionalidades do *Shell*

Os *shells* (*csh, sh, ksh e tcsh*) disponibilizam aos usuários um número de funcionalidades e conveniências operacionais. O conjunto dessas funcionalidades caracteriza uma linguagem de programação completa.

Existe uma grande variedade de ataques usando funcionalidades do *shell* para comprometer a segurança de sistemas UNIX. Passaremos agora a descrevê-las.

3.3.1. Ataques usando a variável PATH

Cada *shell* mantém um *path*, consistindo de um conjunto de diretórios para pesquisar por comandos requeridos pelo usuário. Os elementos desse conjunto de diretórios são consultados, um a um, quando o usuário digita um comando que não contenha um "/" no início do seu nome e que não seja um comando interno ao *shell* ou um alias. Nos *shell* Bourn e Korn, a variável PATH é normalmente preechida pelo arquivo de inicialização. A lista de diretórios dada consiste normalmente de diretórios, separados por ":". A variável PATH no *csh* é inicializada pelo preenchimento da mesma com uma lista, entre parênteses, de nomes de diretórios separados por espaço em branco [30].

Um exemplo típico dessa inicialização é:

```
PATH=.:/usr/bin:/bin:/usr/local/bin          sh ou ksh
set path= ( . /usr/bin /bin /usr/local/bin )  csh
```

No exemplo acima, cada comando é pesquisado nos diretórios corrente (.), */usr/bin, /bin* e */usr/local/bin* nessa ordem. O diretório corrente nunca deve ser incluído no *path*.

Para entendermos o perigo de colocarmos o diretório corrente no *path*, imagine que um atacante crie dois arquivos no diretório */home* de um usuário: um arquivo texto chamado "-i" e um *script shell* chamado "ls", transcrito abaixo:

```
% cat ls
#!/bin/sh
(/bin/cp /bin/sh /tmp/.secret
/etc/chown root /tmp/.secret
/bin/chmod 4555 /tmp/.secret) 2>/dev/null
rm –f $0
exec /bin/ls "$@"
%
```

Esse *script shell* cria uma cópia do *shell*, com o bit SUID ligado no diretório */tmp* com um nome "invisível"[4]. Ele então apaga ele mesmo e executa o comando "ls" com o argumento fornecido pelo usuário.

Um usuário inescrupuloso necessita apenas comunicar ao administrador que existe um arquivo em seu diretório */home* que não pode ser apagado (o arquivo "-i"). O administrador, com privilégios de *root*, entra no diretório do atacante e digita o comando "ls" para visualizar o tal arquivo que não pode ser apagado. Nesse ponto, se o administrador tiver em sua variável PATH, a indicação do diretório corrente no início da lista de diretórios, o "ls" que será executado será o do atacante e não o do sistema, permitindo assim, que seja criado um *shell* de *root* dentro do diretório */tmp* para uso futuro.

[4] Arquivos cujos nomes iniciam com o caracter ". ", normalmente não são listados pelo comando "ls" do UNIX, nem são expandidos pelo *shell*, tornando-se, dessa forma, ocultos.

A chave para prevenir esses tipos de ataques é nunca ter o diretório corrente em seu *path*. Isso é especialmente importante em contas de superusuários! De forma geral, você não deve ter em seu *path* diretórios onde outros usuários possam escrever [30].

Colocar o diretório corrente no final da lista de diretórios do *path* também não é uma boa idéia, pois suponha que um usuário use muito um determinado comando; por exemplo o comando "more". Mas algumas vezes, por um erro de digitação qualquer, o usuário digite "mroe". O atacante pode se aproveitar disso, fazendo um programa Cavalo de Tróia com nome "mroe". Pode demorar algumas semanas ou até meses, mas o comando acaba sendo executado.

3.3.2. Ataques usando a variável IFS

A variável IFS pode ser preenchida para indicar que caracteres separam palavras (similar a opção –F para o *awk*). Essa variável é usada por *scripts shell* que necessitam ler listas de palavras. Por exemplo, você poderia usar o seguinte *script shell* para pegar uma lista de nomes de usuários com seus respectivos diretórios *home* [16].

```
#!/bin/sh
IFS=":"
while read acct passwd uid gid gcos homedir shell
do
        echo $ acct " " $homedir
done < /etc/passwd
```

A funcionalidade do IFS é largamente utilizada por outras ferramentas, tipo *awk* e *perl* e pode causar danos não esperados ao sistema. Selecionando IFS para usar o separador "/", é possível para um atacante fazer com que um arquivo *shell* ou um programa executem comandos do sistema.

Os *shells* mais recentes sempre inicializam os valores de IFS com um conjunto de caracteres normais e não danosos. Contudo, nem todos os *shells* fazem isso. Para determinar se um *shell* está imune a esse problema, podemos executar o seguinte *script*.

```
: Um teste do shell
cd /tmp
cat  > tmp << 'E-O-F'
echo "Seu shell não inicializa variável IFS!"
E-O-F
cat > foo << 'E-O-F'
echo "Seu shell não apresenta o problema."
E-O-F
cat > test$$ << 'E-O-F'
/tmp/foo
E-O-F
chmod 700 tmp foo test$$
PATH=.:$PATH
IFS=/$IFS
export PATH IFS
test$$
rm –f tmp foo test$$
```

No *script* acima, são gerados 3 arquivos visando verificar se um determinado *shell* inicializa ou não a variável IFS. Sendo assim, são gerados, dentro do diretório */tmp*, os arquivos *tmp*, *foo* e *test*. Após isso, o comando *chmod* é usado para alterar as permissões do arquivo *test*, visando torná-lo executável, as variáveis PATH e IFS são inicializadas e exportadas para depois executar o arquivo *test*. Notamos que, se o *shell* não inicializar a variável IFS automaticamente, elá terá o valor "/", o que implicaria dizer que "/" seria o delimitador de palavras na linha de comandos. Como o arquivo *test* executa */tmp/foo*, o *shell* tentará executar primeiro o arquivo *tmp* para depois o *foo*, informando ao usuário que seu shell não inicializa a variável IFS. Caso contrário, o arquivo foo será então executado, informando ao usuário que seu shell não apresenta o problema.

3.3.3. Ataques usando a variável HOME

Outra tática que pode ser explorada, em algumas circunstâncias é inicializar a variável HOME. Normalmente, o *csh* e o *ksh* substituem o valor dessa variável pelo símbolo "~" [15].

Então, se um atacante está habilitado a mudar o valor dessa variável, ele pode tirar vantagem de um arquivo *shell* que use o "~" como representação do diretório *home*.

Por exemplo, se existe uma arquivo *csh* com o *bit* SUID ativo, pertencente ao usuário *root*, que faça referência ao arquivo *"~/.rhosts"* quando um usuário o executa, então é possível subvertê-lo alterando a variável de ambiente HOME antes de executá-lo.

Notamos que o sistema UNIX possui diversas funcionalidades oferecidas por seus *shells* que auxiliam usuários e programadores. No entanto, muitas dessas funcionalidades acabam gerando vulnerabilidades no sistema, causando problemas contra os quais os administradores UNIX travam uma grande batalha.

A administração UNIX é muito complexa e cheia de detalhes. Embora seja bastante difícil tornar um sistema UNIX totalmente invulnerável, podemos reduzir bastante os riscos causados por vulnerabilidades, aplicando procedimentos de controle e conhecendo bem a estrutura de funcionamento do núcleo e das aplicações UNIX.

Capítulo 4
Classificação das Vulnerabilidades em *Sites* Internet

4.1. Introdução

Para que as vulnerabilidades possam ser entendidas e verificadas, precisamos levar em consideração algum tipo de classificação, visando a sistematização do conhecimento, assim como a facilidade de identificação/reconhecimento de uma falha de segurança em um determinado sistema.

Durante essa pesquisa, o que verificamos foi uma grande quantidade de *sites* que disponibilizam alguns mecanismos de exploração de vulnerabilidades, sempre organizados por sistema operacional. Não encontramos nenhuma bibliografia que fornecesse informações, classificadas segundo algum critério, sobre as falhas existentes em aplicações de sistemas, exceto alguns relatórios de organismos internacionais de divulgação de vulnerabilidades, que as descrevem considerando apenas o sistema operacional e a aplicação que possui a falha.

Sendo assim, resolvemos organizar as vulnerabilidades segundo seu ponto de presença. Dessa forma, conseguimos visualizar melhor uma determinada falha, segundo o seu modo de atuação, facilitando dessa forma, a geração de mecanismos que possam proteger os sistemas de alguns ataques.

4.2. As vulnerabilidades

A fim de contextualizar o trabalho, precisamos conceituar **Falha de Segurança** do ponto de vista dos *sites* Internet. As **Falhas de Segurança** podem estar presentes em 4 instâncias bem definidas:

- ✓ Na especificação do TCP/IP;
- ✓ Nas implementações das aplicações TCP/IP;
- ✓ Nas implementações das aplicações do sistema operacional;
- ✓ Nas configurações do sistema operacional.

Sabemos que conceitualmente o projeto do TCP/IP possui algumas falhas. No entanto, uma **Falha de Segurança** só existe no contexto desse trabalho, se a mesma se apresentar como um "fato concreto". Então, uma **Falha de Segurança** deve ser sempre criada por uma implementação. É a implementação que, através de mecanismos de programação que exploram os recursos do hardware e do software, geram possibilidades de se causar uma **Falha de Segurança** em um determinado sistema. Sendo assim, conceituamos:

- ➲ **Aplicação do Sistema Operacional**: Programas que acompanham o sistema operacional e que desenvolvem tarefas especificas. Ex: Pine, Elm, ps, etc.

- ➲ **Aplicação TCP/IP**: Programas implementados sobre a pilha TCP/IP, ou seja, que usam protocolos TCP/IP em sua execução. Ex: Sendmail, nfsd, mountd, tcpd, named, etc.

- ➲ **Falha de Segurança**: É um erro, provocado pela implementação de uma **Aplicação do Sistema Operacional** ou de uma **Aplicação**

TCP/IP, que permite a um usuário remoto ou local realizar atividades normalmente não permitidas para ele, visando parar o sistema e/ou ter acesso a uma determinada informação ou recurso não autorizado.

Sabemos que falhas podem existir desde a concepção de dado projeto; no entanto, nos concentraremos nas falhas após a implementação, pois somente nessa fase é que as falhas podem então ser exploradas e quase sempre podem trazer consequências danosas aos sistemas.

Segundo essas premissas, passamos agora a descrever uma proposta de classificação para as vulnerabilidades encontradas em *sites* Internet.

Considerando que tais vulnerabilidades somente podem estar presentes em implementações de uma determinada aplicação, temos quatro grandes classes:

- As vulnerabilidades impostas por usuários - USERWARE
- As vulnerabilidades em implementações de aplicações TCP/IP
- As vulnerabilidades em implementações de aplicações do S.O.
- As vulnerabilidades em configurações do S.O.

As vulnerabilidades em aplicações TCP/IP envolvem implementações de *sendmail, pop3, telnet, ftp, ping, http*, etc. Enquanto que os problemas com aplicações do S.O. envolvem implementações de *mount, pine, elm, ps, fdformat,* etc. Por outro lado, existem problemas de segurança que são proporcionados por erros de configuração em sistemas operacionais.

O que notamos é que algumas aplicações TCP e do SO possuem vulnerabilidades que são exploradas apenas se existirem problemas de configuração do S.O. Portanto, existe uma interdependência entre as classes de vulnerabilidades propostas. A figura 4.1 ilustra essa interdependência.

<table>
<tr><td colspan="6">USERWARE</td></tr>
<tr><td colspan="6">APLICAÇÃO</td></tr>
<tr><td>Spoof</td><td>Flood</td><td>TCP sequence</td><td>Buffer overflow</td><td>Race condition</td><td>Passagem de parâmetros</td></tr>
<tr><td colspan="6">CONFIGURAÇÃO</td></tr>
<tr><td colspan="2">Tabela de partição</td><td colspan="2">Permisão de arquivos</td><td>Flags de inicialização</td><td>Rucursos sub-utilizados</td></tr>
<tr><td colspan="6">SISTEMA OPERACIONAL</td></tr>
<tr><td colspan="6">HARDWARE</td></tr>
</table>

Figura 4.1: Estrutura de Classificacao das Vulnerabilidades.

Na figura 4.1, observamos as quatro classes de vulnerabilidades propostas e suas sub-classes. A classe USERWARE assume como vulnerabilidades questões como senhas de usuário fáceis, contas de usuários esquecidas abertas, etc. Após esse nível, encontramos a classe APLICAÇÃO com suas sub-classes envolvendo problemas como *Buffer Overflow, Race Condition*, Passagem de parâmetros, sendo essas comuns as aplicações do sistema operacional e as aplicações TCP. As sub-classes *spoof, flood* e TCP *Sequence* são inerentes apenas à aplicações TCP, pois envolvem manipulação de dados TCP através da implementação de suas aplicações. A última classe na estrutura de classificação mostrada na figura 4.1 é a classe CONFIGURAÇÃO que também possui suas sub-classes. Cada uma dessas sub-classes será detalhada a seguir:

Classe APLICAÇÃO

a) ***Buffer Overflow***: É uma vulnerabilidade que consiste em uma aplicação gerar um estouro na area de armazenamento auxiliar de processamento em tempo de execução. Isso ocorrendo, apontadores da aplicação são então liberados e devidamente tratados por funções especiais (*exploits*), visando executar programas não autorizados com privilégios especiais.

b) ***Race Condition***: A grande maioria dos sistemas operacionais multiusuario e multitarefa, ao executar algumas aplicações, geram arquivos temporários que, em primeira instância possuem privilégios de *root* (usuário administrador do sistema). Determinados programas (*exploits*) usam essa vulnerabilidade para tentar (até conseguir), ligar um desses arquivos com um arquivo de um outro usuário sem privilégios. Uma das vezes em que o *exploit* executar e conseguir "tomar" um arquivo com direitos e permissões de *root*, então o usuário que estiver executando o *exploit* pode passar a ter os mesmos privilégios.

c) ***Passagem de Parâmetros***: É uma vulnerabilidade que permite a uma aplicação cliente passar parâmetros a um determinado servidor, de forma a driblar a sequência normal de execução de determinado *daemon* por parte desse servidor. Com isso, usuários podem manipular arquivos com permissões de *root* ou mesmo obter os privilégios de outros usuários.

Além dessas sub-classes que são inerentes a aplicações do sistema operacional, existem ainda outras que são inerentes apenas a aplicações TCP/IP. São elas:

a) ***Spoof***: É uma vulnerabilidade existente em servidores de nomes e consiste em fazer uma máquina remota ser reconhecida em uma rede local como uma máquina válida e confiável. Considerando que hoje os *sites* Internet têm se utilizado muito de *firewalls* para proteção de suas redes internas e considerando também que a maioria dos *firewalls* atuam filtrando pacotes e serviços, essa vulnerabilidade permite ultrapassar as barreiras do *firewall*.

b) ***Flood***: Considerando que o protocolo TCP é orientado a conexão, sabemos que uma máquina que solicita uma conexão TCP a outra, transmite um *SYN request* para a maquina a conectar. Essa maquina então responde a chamadora com um *SYN ACK*. Sendo assim, essa vulnerabilidade permite a um usuário reduzir a performance de uma maquina ou mesmo para-la por

completo, através do envio de diversos pacotes TCP com *SYN request*, visando fazer com que a máquina destino fique tão ocupada em responder aos pacotes que pare então de responder às demais solicitações de serviço ou mesmo que estoure seu *buffer* de execução.

c) ***TCP Sequence***: É uma vulnerabilidade existente em implementações TCP. Sabe-se que uma máquina envia sempre uma mesma sequência de predicados TCP. A vulnerabilidade permite que um usuário gere, a partir de uma maquina qualquer na Internet, uma sequência TCP igual àquela da máquina que se quer atacar. Isso permite, juntamente com o *Spoof* e o *Flood*, um ataque que iluda completamente *firewalls* baseados em filtros de IP.

Classe CONFIGURAÇÃO

a) ***Tabela de Partição***: É a vulnerabilidade gerada quando o sistema operacional é instalado em uma só partição, ou seja todos os sistemas de arquivos são montados em cima de apenas uma partição no disco. Isso facilita bastante a *Race Condition* em alçumas aplicações TCP/IP ou do S.O., pois somente é possível ligar dois arquivos que estejam na mesma partição. Se um arquivo foi gerado no sistema de arquivos ***/tmp*** e um programa *exploit* quizer ligá-lo com um arquivo localizado no ***/home*** e esses dois sistemas de arquivos estão em partições diferentes, essa ligação nunca poderá ser feita, reduzindo a possibilidade do sistema estar vulnerável a *Race Conditions*.

b) ***Permissão de Arquivos***: Essa vulnerabilidade habilita um usuário a ler, escrever ou executar um arquivo proibido. Sistemas com essa vulnerabilidade podem permitir um grande número de ataques.

c) ***Flags de Inicialização***: É uma vulnerabilidade causada pela habilitação de opções na inicialização das aplicações, de maneira que as mesmas possam então fornecer dados importantes sobre o sistema operacional e sua configuração. Um exemplo típico disso é o *sendmail* inicializado com a opção

-d (*debug*) que permite a um usuário, na porta 25 do TCP, verificar os *usernames* dos usuários do sistema, dentre outras coisas.

d) ***Recursos Sub-utilizados***: São recursos do sistema que, muitas das vezes são inicializados por *default*, porém não estão sendo realmente utilizados pelos usuários do mesmo. Esses recursos, muitas vezes podem trazer consequências graves para a segurança do *site* e como muitas vezes não são utilizados ou podem ser substituídos por outros, podem não ser inicializados. Exemplos disso: o tftp, que em alguns sistemas é inicializado e não é utilizado, o *rlogin*, o *telnet*, o *sendmail*, o *talk*, etc.

Notamos que as vulnerabilidades encontradas em *sites* Internet podem ser classificadas segundo o modelo proposto neste capítulo. As formas como essas vulnerabilidades são exploradas não foram caracterizadas nessa classificação, pois podemos explorar uma determinada vulnerabilidade através da combinação de uma série de técnicas tanto locais como remotas. Optamos então por caracterizá-las e descrevê-las como formas de ataques

Capítulo 5
Considerações sobre algumas formas de ataque

5.1. Introdução

Existem muitas formas de protegermos *sites* Internet contra a ação de invasores. Contudo, a cada dia surgem novas aplicações que trazem funcionalidades indispensáveis a um sistema. Essas ferramentas, como programas, podem e vêm sendo depuradas por invasores, visando encontrar vulnerabilidades que possam auxiliá-los na obtenção de direitos de acesso a um determinado *site*.

As vulnerabilidades podem ser exploradas por atacantes de forma local ou de forma remota, ou seja, um atacante pode possuir uma conta de acesso a um shell do sistema e a partir daí desencadear o ataque, ou pode ainda não possuir nenhuma conta de acesso, portanto não ser um usuário do sistema e mesmo assim conseguir aplicar o ataque.

Neste capítulo, faremos algumas considerações sobre as formas mais frequêntes de ataques.

5.2. Quem ataca: conceito de *hacker*

Ataques a sistemas são normalmente realizados por *hackers* ou *crackers*. O termo *hacker* já se encontra hoje associado a pirata digital, invasor de sistemas e criminoso. Mas nem sempre foi assim. Segundo "O Novo dicionário de *Hackers*" (The New *Hackers*'s Dictionary) [31], o *hacker* é uma pessoa que gosta de explorar os detalhes dos sistemas e descobrir como obter o máximo de sua capacidade, em oposição à maioria dos usuários, que preferem aprender apenas o mínimo necessário. Outra definição: aquele que programa entusiasticamente (até de forma obsessiva), ou que prefere programar a teorizar sobre programação. Seguindo rigorosamente o vocabulário do meio, o *hacker* que se dedica a roubar arquivos ou destruir dados ganha o nome de *cracker*. Esses sim, são os *hackers* perigosos.

No início, de fato, os *hackers* eram apenas curiosos que adoravam "debulhar" sistemas para descobrir suas falhas de segurança. Se penetravam num servidor, faziam como se estivessem passeando num prédio sem serem vistos. Não tocavam em nada, apenas observavam. Mas logo apareceram *hackers* que não resistiram às tentações que surgiam em cada servidor invadido e começaram a destruir ou roubar dados. Suas mais célebres proezas acabaram popularizando o nome como símbolo de invasão criminosa e pirataria digital.

5.3. Estratégias de Ataque

Os primeiros alvos do ataque de um hacker iniciante são normalmente os provedores Internet ou servidores de grandes instituições como as Universidades. Os invasores precisam de uma plataforma para desenvolver seus ataques sem deixar pistas. Seu primeiro objetivo é sempre o arquivo de senhas de acesso. De posse desse arquivo e de um programa que permite

reconhecer as senhas encriptadas através de um processo de comparação[1], o hacker já pode garantir a entrada na rede sem pagar, jogando a conta em cima de outro usuário. Se conseguir capturar de 40 a 50 senhas, pode usar uma hora por dia de cada usuário, que dificilmente será descoberto.

Mas isso ainda é pouco. Logo que entra no sistema, o *hacker* procura adquirir o *status* de superusuário do sistema, ou *root*, o administrador do sistema. Com isso, ele pode fazer o que quiser, desde criar usuários fantasmas, que garantam seu acesso futuro, a excluir outros usuários, criar diretórios ocultos para esconder seus arquivos e muito mais. Para suas proezas, os *hackers* iniciantes dependem muito de programas já prontos que permitem identificar senhas ou assumir controle de sistemas.

As atividades dos *hackers* podem ser das mais inofensivas - por exemplo, fazer modificações numa *home-page* - até as mais nocivas, como tirar provedores do ar, apagar arquivos ou discos rígidos inteiros, roubar números de cartão de crédito armazenados em lojas virtuais de comércio eletrônico, penetrar em computadores de instituições como a NASA ou a CIA, etc.

Mesmo quando uma invasão é descoberta, nem sempre o servidor violado fica a salvo. Se não for feita uma rigorosa limpeza, podem ficar no computador *sniffers*, programas deixados pelo invasor que registram todo o movimento de pacotes de informação e que podem continuar roubando senhas e qualquer outro tipo de informação.

A substituição de *home-pages* pode até ser politicamente correta - como a dos *altavistas* Portugueses e Timonenses que invadiram a página do Ministério das Relações Exteriores da Indonésia para denunciar a ocupação

[1] Programas de reconhecimento de senhas baseiam-se em um dicionário de palavras onde é aplicado o algoritmo de criptografia conhecido, gerando sequencias de caracteres equivalentes as senhas contidas no dicionário. Feito isso, é realizada uma comparação de sequencias de caracteres entre o arquivo de senhas da máquina atacada e o arquivo gerado pelo programa de reconhecimento de senhas.

ilegal que este país exerce em Timon Leste - ou até francamente hilariante, como a que um grupo de *hackers* fez no site da seita Portal do Paraíso, substituindo o site inteiro da *High Source*, empresa de criação de *home-pages* que a seita administrava, por um outro que seguia o lema "Nós nos matamos de trabalhar por você." Podemos encontrar exemplos destas invasões em "http://bandit.pangeia.com.br/pm/". No Brasil, o caso mais recente deste tipo de atividade foi a substituição em janeiro deste ano da *home-page* da Biblioteca Nacional pela página do Comando Maconha Brasil. No anexo 1, encontramos alguns exemplos de páginas web alteradas por hackers nos últimos anos.

Outra metodologia muito usada é a "engenharia social", quando o *hacker* engana funcionários de provedores e os leva involuntariamente a passar-lhe informações através de um papo bem construído.

Os *hackers* mais sofisticados também são *phreakers*[2], que conhecem profundamente o sistema de telefonia internacional e conseguem fazer ligações internacionais sem pagar. Muitos destes preferem estabelecer a base de seus ataques no exterior, para despistar ainda mais possíveis rastreadores. O supra-sumo destes piratas digitais são os que já adquiriram conhecimentos na área de satélites. Pode parecer coisa de ficção científica, mas existem *hackers* capazes de interferir no funcionamento de satélites, mudar sua órbita e provocar a interrupção de seu fluxo de informação. Pelo menos em teoria, já há conhecimento fora da comunidade científica suficiente até para interferir em emissões de televisão via satélite, pondo outras imagens no ar.

[2] Phreakers: São hackeres que atuam em sistemas telefônicos.

5.4. Mecanismos usados em ataques

Um atacante pode, dentre outras coisas, verificar quais as características do *site* que deseja atacar, visando obter informações detalhadas sobre características de *hardware* e *software*. Considerando que o ataque pode ocorrer de forma remota ou local, o atacante deve visar, primeiramente conseguir acesso local. Esse tempo existente entre o ataque local e o remoto caracteriza a aplicação de *exploits* remotos para a obtenção de senhas locais.

5.4.1. Ataques Locais

Para produzir um ataque local, um atacante deve possuir um acesso regular à máquina, ou seja, ele deve possuir uma senha de acesso a uma conta de um usuário válido para o sistema alvo. Possuindo esse acesso, o atacante deve conhecer bem o sistema operacional local e suas vulnerabilidades para poder então separar os programas *exploits* capazes de explorar algumas dessas vulnerabilidades. Ataques locais normalmente exploram vulnerabilidades classificadas como **Aplicações TCP** ou **Aplicações S.O.**, como mostramos no capítulo 4.

Um atacante pode utilizar programas conhecidos como *exploits* para desenvolver um ataque a uma aplicacão do sistema operacional como o pine, por exemplo, usando uma vulnerabilidade da sub-classe *race condition*. *Race condition* consiste em permitir que um arquivo de um usuário, gerado por uma aplicação, seja ligado a um outro arquivo (o do atacante), visando obter os privilégios daquele usuário. Se o usuário escolhido para o ataque for o administrador do sistema (*root*), então o atacante terá conseguido seu objetivo: conseguir privilégios máximos sobre o sistema, estabelecendo ali sua base de ataques. Por exemplo, ao executar algumas versões do programa pine (leitor de mails do UNIX), é gerado um arquivo de controle dentro do diretório /tmp. Esse arquivo pertence ao usuário que está usando o

pine naquele instante e possui permissões de leitura e gravação para todos os usuários do sistema. Um atacante pode então gerar um exploit capaz de verificar se existe aquele arquivo, alterá-lo de forma a conter a sequência de caracteres "+ +"[3] e então ligá-lo a um arquivo chamado .rhosts dentro da conta do usuário que estiver utilizando o pine.

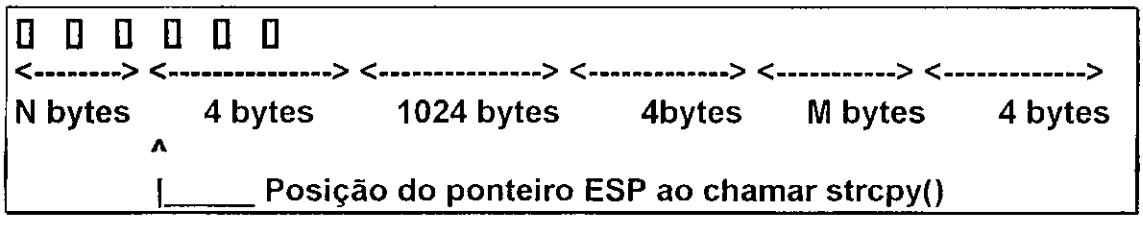
Além de programas que exploram vulnerabilidades da classe *race condition*, existem programas que exploram a mais classe comum das vulnerabilidades, a ***Buffer Overflow***. O problema de *buffer overflow* vem do fato de que programadores constroem códigos com passagem de parâmetros do tipo:

```
void parse(char *arg) {
        char param[1024];
        int localdata;
        strcpy(param,arg);
        .../...
        return;
}
main(int argc, char **argv) {
        parse(argv[1]);
        .../...
}

ou

main(int argc, char **argv) {
        char param[1024];
        int index=0;
        while ((param[index++]=getchar()) != EOF);
        .../...
}
```

[3] A sequência de caracteres "+ +" dentro do arquivo .rhosts localizado do diretório **home** de um usuário, habilita a relação de equivalência entre máquinas, ou seja, se o usuário **jose** possui um arquivo **.rhost** com o conteúdo "+ +" em seu diretório **home**, ele estará habilitado a usar o comando **rlogin**, a partir de qualquer máquina na rede, para logar-se em sua conta sem a necessidade de fornecer uma senha.

Como vemos, nenhuma verificação é feita no tamanho da cadeia de caracteres. Programadores costumam considerar que ninguém irá fornecer um nome de arquivo com mais de 1024 caracteres e se isso acontecer, o usuário receberá uma mensagem de *segmentation fault*.

Em sistemas Linux (ou em outros sistemas baseados no Interx86), a pilha de execução caminha de cima para baixo. Quando um programa chama uma função, o endereço de retorno é retirado da pilha e o ponteiro (ESP) é decrementado de 4 (32 bits). Então, se uma função define variáveis locais (tipo "param" no nosso exemplo), o vetor é definido para o endereço de retorno da função. Quando o strcpy() executa no primeiro programa de nosso exemplo, a pilha estará assim:



Se um atacante produzir um código de execução de um *shell* com exatamente 1024 bytes e adicionar 4 bytes ao ponteiro ESP antes do código do programa executar, então a função retornará exatamente para a posição onde está o código do shell, executando-o.

Se a aplicação que possui esse código executar com privilégios de *root*, o código do shell do atacante também será executado com os mesmos privilégios, fornecendo acesso de superusuário ao atacante.

Ataques locais possuem uma grande variedade de formas. Por exemplo, imagine se um usuário deixa, mesmo que por alguns instantes, sua conta aberta em um terminal e vai tomat um cafezinho. Um atacante pode executar a seguinte sequência de comando:

```
cp /bin/sh /tmp/.aaa
chmod 7455 /tmp/.aaa
```

Isso deve gerar uma cópia do arquivo **/bin/sh** dentro de um arquivo oculto chamado **.aaa** no diretório **/tmp**. A segunda linha de comando ativa o SUID do arquivo gerado. Como a cópia foi feita a partir da conta de um usuário, o arquivo oculto gerado será de propriedade daquele usuário. Com o SUID ligado, o atacante poderá, mais tarde, executar o arquivo **.aaa**, obtendo todos os privilégios do *shell* daquele usuário.

5.4.2. Ataques Remotos

Ataques remotos podem visar paralizar alguns serviços do sistema ou simplesmente obter o acesso local. Alguns ataques que visam paralizar serviços são clássicos, como por exemplo ataques da sub-classe de vulnerabilidades *Flooding*. Sistemas que apresentam essa vulnerabilidade podem ter suas atividades totalmente paralizadas. A técnica de ataque que explora essa vulnerabilidade se sustenta no fato de que toda requisição de conexão TCP só é estabelecida após o *three-way-handshake*. Se o atacante enviar uma sequência muito grande de requisições a uma máquina vulnerável, a mesma irá armazenar essas requisições em sua pilha e ficará tão ocupada em responder ao atacante, que não "terá tempo" de responder a serviços básicos de sua conexão local em tempo hábil. Isso provocará uma sensação de travamento para os usuários da máquina atacada. Em versões mais antigas de implementações TCP, pode acontecer um estouro da pilha da máquina atacada, resultando no real travamento da mesma.

Além disso, existem ataques baseados em *Spoofing*. Neste tipo de ataque, o intruso transmite pacotes a partir da rede externa que fingem ser originários de uma máquina interna - os pacotes falsificados contém o endereço IP fonte que especifica uma máquina interna da rede. O intruso, então, espera que

uma verificação simples de endereço seja feita, de modo a permitir que pacotes de máquinas internas confiáveis sejam aceitos ao mesmo tempo em que pacotes de outras máquinas sejam descartados.

Uma vez tendo sido autenticado pelo sistema, o intruso pode executar programas *exploits* visando explorar vulnerabilidades no sistema e está configurado o ataque (na verdade, o IP *spoofing* não é o ataque, mas é apenas um passo no sentido de configurá-lo). O Spoofing é muito usado por atacantes contra *sites* que possuem tecnologia *firewall* instalada.

Uma boa técnica para detectar um ataque como este é descartar pacotes que chegam pela interface externa do roteador com endereços IP fonte que especificam máquinas internas.

Um outro ataque remoto muito comum é aquele que usa o *TCP sequence number*. Para entendermos melhor isso, vamos estudar rapidamente a sequência usado no *three-way-handshake*. Supondo que a máquina cliente A quer falar com a máquina servidora B. Ela transmite a seguinte mensagem:

A -> B: SYN, ISSa

Isto é, ela transmite um pacote com o SYN (Synchronize sequence number) bit ligado e um número sequencial inicial ISSa. Sendo assim, B responde com:

B -> A: SYN, ISSb, ACK(ISSa)

Notamos que B também envia para A seu número sequencial inicial e um ACK. A entao conclui o handshake com:

A -> B: ACK(ISSb)

Se um atacante X abrir uma conexão com uma porta qualquer de uma máquina B, ele pode então obter o número sequencial emitido por B (ISSb). Ele pode então se fazer passar por A (usando *spoofing*) e transmitir:

A -> B: SYN, ISSx

A máquina B então responde ao SYN original de X:

B -> A: SYN, ISSb, ACK(ISSx)

Assim, a máquina B aceita a conexão da possível máquina A, quando na verdade está aceitando conexões de X. Essa técnica é muito usanda para "driblar" alguns mecanísmos de proteção baseados em *wrappers*[4].

[4] Wrappers: Mecanismos de software usados para filtrar alguns serviços como: telnet, ftp, rlogin, etc.

Capítulo 6
O Ambiente Groove

6.1. Introdução

Neste capítulo, será apresentado um ambiente de verificação da segurança de *sites* Internet denominado Groove, incluindo a descrição funcional de suas ferramentas, seus objetivos, justificativas, benefícios, arquitetura e protocolos.

O Groove foi concebido considerando a grande carência de profissionais que ocupam funções de administração técnica de *sites* Internet e que incorporam atividades de manutenção da segurança de seus sistemas.

O ambiente aqui descrito não visa tornar um *site* seguro, mas sim fornecer subsídios a administradores no intuito de indicar onde e porque seus sistemas encontram-se vulneráveis.

O Groove verifica *sites* Internet através de uma sequência de testes que envolve duas fases:

Fase A: Aplicação de testes através da execução de scripts que avaliam a versão da aplicação com o conhecimento prévio das versões onde as vulnerabilidades se apresentam;

Fase B: Aplicação de programas que exploram as vulnerabilidades, independente da versão da aplicação encontrada no sistema alvo. Essa fase visa aumemtar o nível de confiabilidade dos testes, considerando que não podemos prevêr se o sistema alvo dos testes já foi ou não invadido

anteriormente e por isso, não podemos confiar completamente nas informações fornecidas pelo sistema operacional e pelas aplicações durante a fase A.

Notamos durante a pesquisa, que existem inúmeras vulnerabilidades que surgem a cada dia e que nem sempre aparecem acompanhadas dos respectivos programas *exploits*. Normalmente a divulgação das vulnerabilidades acontece através de boletins informativos, onde são apresentados apenas dados sobre a descrição do problema, o impacto, a versão da aplicação que apresenta a vulnerabilidade e possíveis soluções. Com essas informações, podemos facilmente gerar *scripts* que testem um sistema a procura de versões problemáticas de algumas aplicações, caracterizando assim, a importância da **fase A** de nossos testes.

Além disso, o Groove fornece informações instrucionais sobre as vulnerabilidades encontrados nos sistemas, visando mostrar a administradores e desenvolvedores de aplicações como funcionam os ataques às vulnerabilidades de seus sistemas e aplicações.

O ambiente proposto possui cinco ferramentas com funcionalidades distintas:

- ✓ **PVU** (Protocolo de Verificação de Usuário): Protocolo do Groove que possibilita a autenticação de usuários para a utilização dos serviços do Groove;

- ✓ **SPV** (Serviço de Pesquisa de Vulnerabilidades): utilizado para possibilitar consultas "*on-line*" de vulnerabilidades em sistemas operacionais específicos, utilizando para isso uma base de dados mantida em um certo servidor;

- ✓ **VSI** (Verificador de *Sites* Internet): aplicação utilizada pelo cliente (administrador do *site*), visando aplicar uma sequência de testes no sistema alvo;

✓ **Imaged** (Imagem daemon): aplicação que deve ser instalada no sistema alvo da verificação e que permite ao administrador monitorar seu *site* e detectar possíveis ataques;

✓ **GBT** (Gerenciador da Base de Testes): aplicação responsável pela manipulação das bases de testes do ambiente Groove.

Essas ferramentas serão funcionalmente detalhadas na seção 6.3. Antes apresentaremos as bases de dados utilizadas pelo ambiente Groove.

6.2. As Bases de Dados Groove

O Groove conta com duas bases de dados bem definidas:

- Base de Dados de Testes e Vulnerabilidades;
- Base de Dados de Usuários.

6.2.1. Bases de Dados de Testes e Vulnerabilidades

Essa base de dados garante toda a funcionalidade do ambiente Groove. Ela armazena informações sobre *scripts* de testes, vulnerabilidades e *exploits*, seguindo a classificação proposta no capítulo 4.

Essa base de dados contém elementos que permitem vários tipos de consultas, utilizando diversas chaves de pesquisa sempre compostas pela união de seus campos.

Notamos que cada campo armazena uma informação necessária para os serviços Groove. Passaremos agora a descrever cada um desses campos:

a) **SO**: Indica o nome do Sistema Operacional que apresenta a vulnerabilidade. Esse campo pode apresentar dados da forma: Linux, Solaris, AIX, etc;

b) **VSO:** Indica a Versão do Sistema Operacional que apresenta a vulnerabilidade;

c) **CLASS:** Identifica a qual classe pertence a vulnerabilidade. Esse campo pode ter os seguintes valores: Aplicação, Configuração e Userware;

d) **SUBCLASS**: Identifica a qual sub-classe pertence a vulnerabilidade. Esse campo contém os nomes das sub-classes comentadas no capítulo 4: Buffer Overflow, Race Condition, Passagem de Parâmetros, TCP Sequence, Spoof, Flood, Tabela de Partição, Permissão de Arquivos, Flags de Inicialização e Recursos Sub-utilizados;

e) **IDENT:** Indica o nome da aplicação que apresenta a vulnerabilidade. Esse campo pode conter informações do tipo: sendmail, telnetd, ps, etc;

f) **SEQ:** Indica qual o número sequencial da vulnerabilidade em relação a aplicação indicada no campo IDENT.

g) **VER:** Indica a versão da aplicação que apresenta a vulnerabilidade.

h) **FUN:** Indica um apontador para a função de testes. Esse campo deve conter o nome do arquivo de teste a ser aplicado, ou seja, o nome do *exploit* ou do script de teste associados a vulnerabilidade;

i) **TIPO:** Indica o tipo de teste especificado no campo FUN. Esse campo deve conter um dos seguintes valores: 0 se o campo FUN contiver o

nome de um script de teste e 1 se o campo FUN contiver o nome de um *exploit*;

j) **TEXT:** indica um apontador para a explicação da vulnerabilidade. Esse campo deve conter o nome de um arquivo texto que contenha um texto explicativo sobre a vulnerabilidade, incluindo uma possível solução para o problema;

l) **FORM:** Indica a forma de exploração da vulnerabilidade. Esse campo deve conter valores do tipo: local, remota, ambas;

6.2.2. Bases de Dados de Usuários

Essa base de dados tem a função de armazenar as informações de cadastro dos usuários do Groove, visando a autenticação dos mesmos antes que os serviços lhes sejam disponibilizados. A base de dados de usuários contém 11 campos assim organizados: NOME, TELEFONE, FAX, CONTATO, ENDEREÇO, ENDEREÇO ELETRÔNICO, PASSWORD. Todos esse campos são preenchidos pelo usuário durante o seu cadastramento feito pelo PVU - Protocolo de Verificação de Usuários, que será descrito adiante. A base de dados de usuários será usada pelo **PVU - Protocolo de Verificação de Usuários** do Groove, visando autenticar os seus usuários e manter sempre atualizados os dados dos mesmos, visando futuros contatos.

6.3. Arquitetura e Serviços

O Groove, como descrito anteriormente, possui cinco ferramentas e sua arquitetura depende diretamente da integração de seus serviços, visando facilitar o alcance de seus objetivos.

A Figura abaixo ilustra a arquitetura Groove.

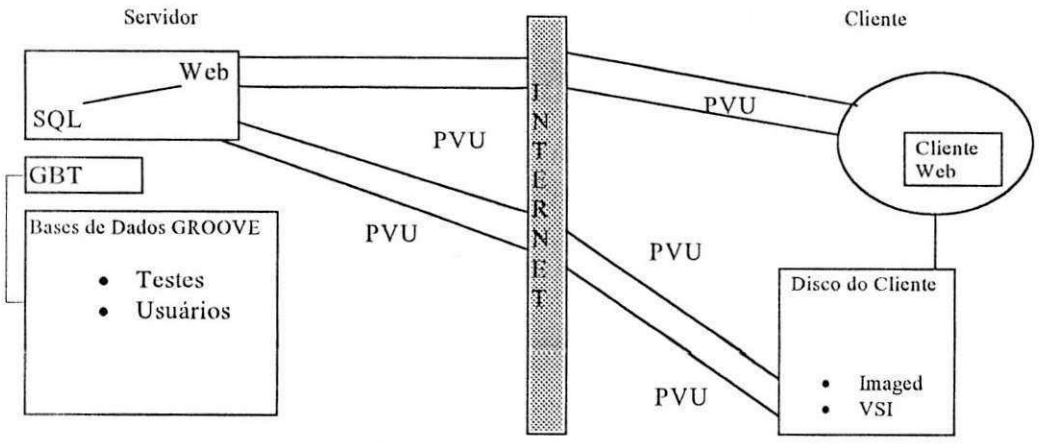


Figura 6.1: Arquitetura GROOVE

Como mostra a figura 6.1, o ambiente Groove é composto de cinco ferramentas as quais o usuário pode ter acesso via Internet seja através de applets java, seja através de download de pacotes para a instalação em seu sistema local.

O Groove possui suas funções centralizadas em operações de consultas a bases de dados gerenciadas por um servidor SQL. Todas as interações entre as ferramentas e serviços do Groove são feitas através de manipulações das bases de dados. Um usuário pode se cadastrar no Groove usando o PVU - Protocolo de Verificação de Usuários, sendo assim incluído na base de dados de usuários; depois disso, o mesmo pode solicitar uma consulta à base de dados de testes e vulnerabilidades, o que acionará novamente o PVU para autenticar o usuário através de uma consulta prévia ao seu cadastro. Além disso, existem ainda ferramentas como o VSI e o Imaged que devem ter seus códigos transferidos do servidor Groove para a máquina do cliente para a sua instalação e execução. Essa transferência é feita via Internet, utilizando o protocolo ftp com chamadas em *background.* Existe disponível ainda o GBT - Gerenciador da Base de Testes, que disponibiliza funções de inclusão, exclusão, alteração e pesquisa na base de dados de testes e vulnerabilidades do Groove.

6.3.1. Acessando o Groove

O cliente pode apontar seu navegador *web* para um servidor Groove[1] para ver a *home-page* inicial do Groove, mostrada na figura 6.2.

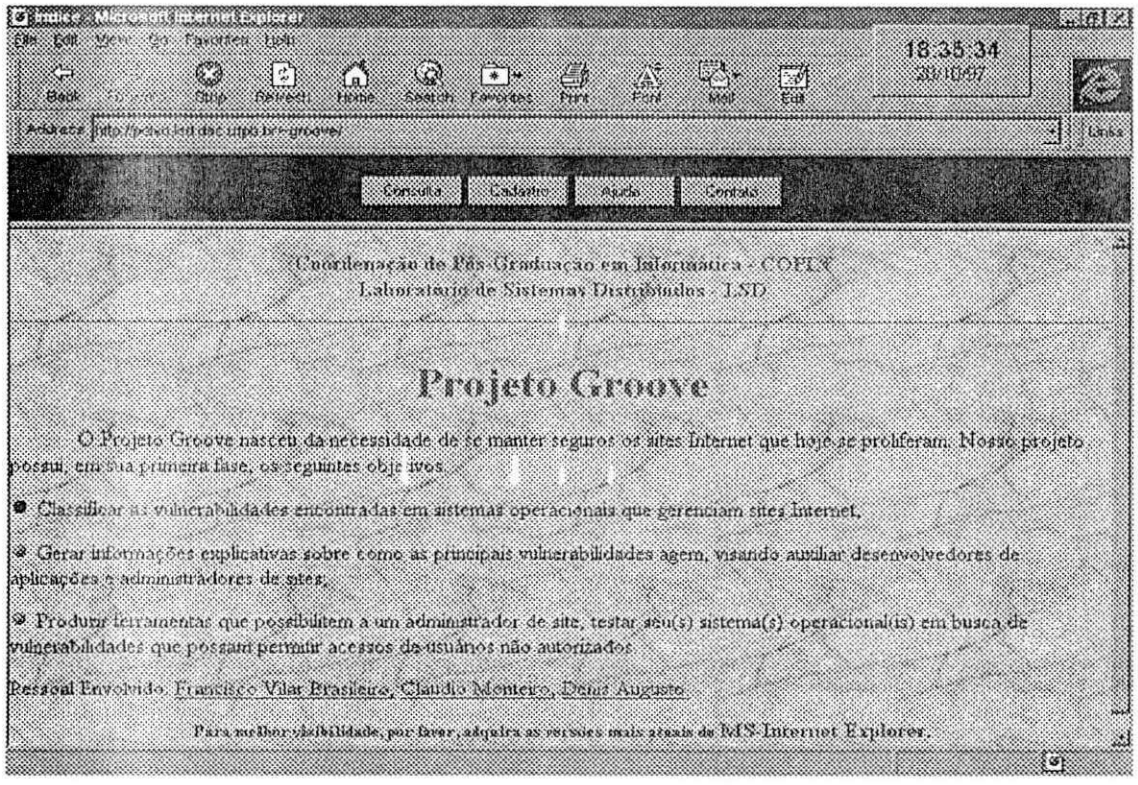


Figura 6.2: Home-page inicial do Groove

Na *home-page* inicial, encontramos quatro botões cada qual com sua funcionalidade que descreveremos a seguir.

➢ **Contato**: O usuário pode optar por esta funcionalidade se necessitar saber como entrar em contato com a equipe do Projeto Groove. Ao escolher essa opção, o usuário terá informações de contato com a equipe de suporte do projeto, como endereço eletrônico, telefone, fax, endereço convencional, etc.

[1] Atualmente existe um disponível em http://polvo.lsd.dsc.ufpb.br/~groove/

➢ **Ajuda**: Essa funcionalidade disponibiliza toda a ajuda necessaria para que um usuário possa utilizar os serviços do Groove.

➢ **Cadastro**: Este botão aciona o PVU - Protocolo de Verificação de Usuários do Groove. Se o usuário ainda não estiver cadastrado, ele deve escolher essa opção para realizar seu cadastro.

➢ **Consulta**: Se o usuário já estiver cadastrado, ele deve escolher essa opção para então ter acesso aos serviços do Groove, após sua autenticação feita pelo PVU.

6.3.2. Protocolo de Verificação de Usuário - PVU

O Protocolo de Verificação de Usuário visa garantir que um dado usuário esteja devidamente autorizado a usar os serviços Groove.

A comunicação entre um cliente e um servidor executando serviços Groove, utiliza os protocolos TCP/IP mais comuns, assim como o HTTP.

Para que essa comunicação seja iniciada, o Groove precisa saber "quem" está usando o sistema e para isso usa um cadastro feito via *web* contendo as informações sobre o usuário. Nesse cadastro, o usuário fornece informações pessoais como nome, endereço, e-mail, além de escolher uma senha de acesso ao Groove. Esse cadastro será armazenado na base de dados de usuários do Groove, para aguardar a liberação de uso que será feita manualmente pela equipe de suporte do Groove.

Se o usuário não estiver cadastrado no Groove, ele deve então escolher o botão cadastrar da *home-page* inicial do Groove, visando acionar o PVU para então poder fazer o seu cadastro. Um exemplo desse cadastro é mostrado na figura 6.3.

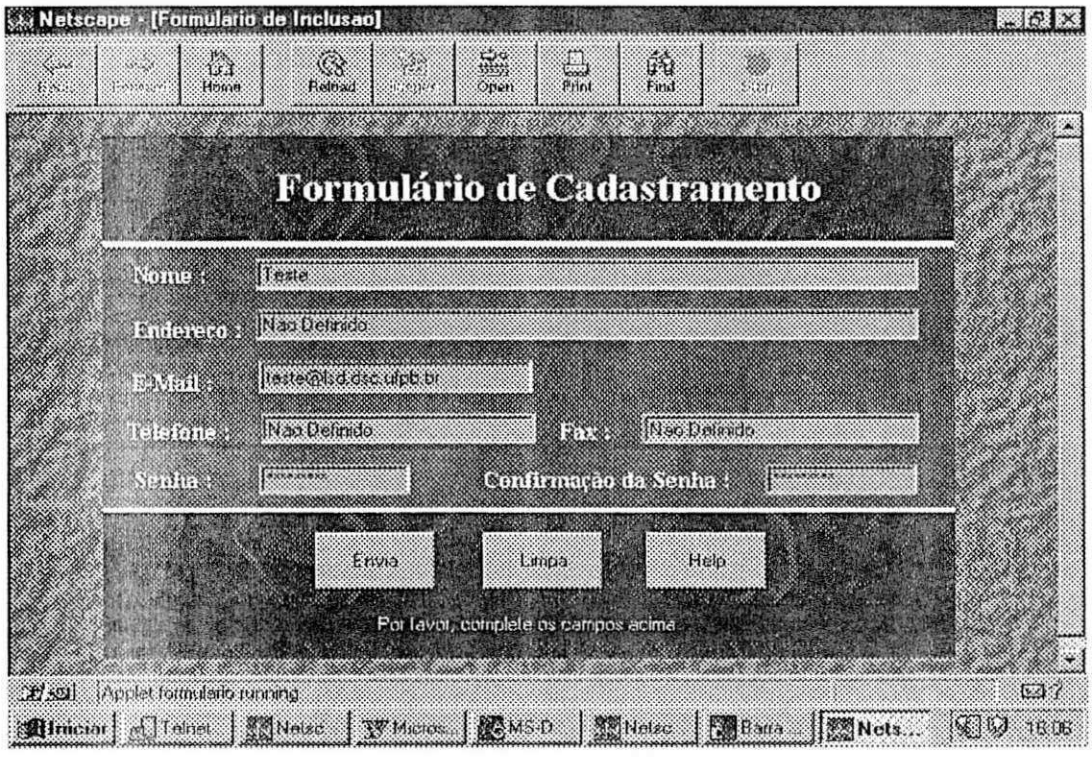


Figura 6.3: Exemplo da tela de cadastro de usuário

Por outro lado, se o usuário já estiver cadastrado, o PVU informa essa situação ao usuário, indicando que o mesmo já pode utilizar os serviços e ferramentas do Groove.

Após liberado o cadastro do usuário, o mesmo pode ter acesso aos serviços do Groove, escolhendo um dos botões disponíveis na *home-page* inicial do Groove. Escolhendo o botão **consulta** ou o botão **download**, o usuário terá acesso ao SPV ou ao VSI e Imaged respectivamente, desde que forneça a senha escolhida juntamente com seu endereço eletrônico ao PVU, visando sua autenticação. Um exemplo desse procedimento é ilustrado na figura 6.4.

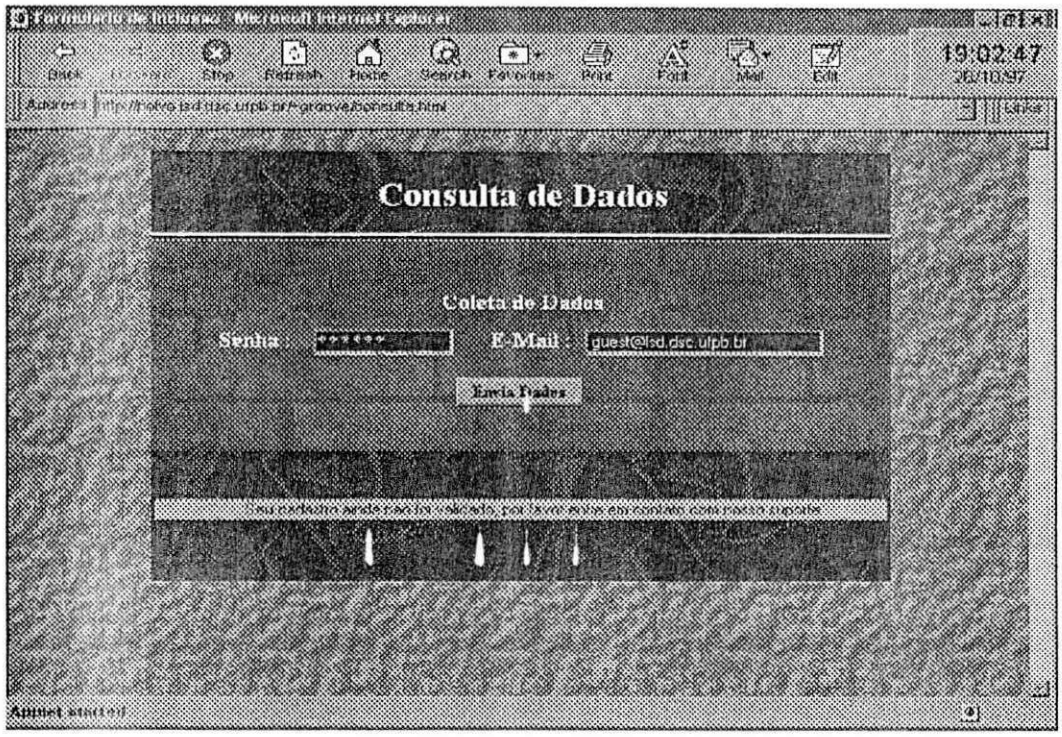


Figura 6.4: Exemplo de Verificação do PVU

Notamos que, ao solicitar um serviço Groove, o usuário, se já estiver cadastrado, deve fornecer seu código de acesso, composto de sua senha e seu endereço eletrônico. Essas informações devem ser devidamente criptografadas pelo PVU, visando a tranferência segura pela rede.

6.3.2. O Serviço de Pesquisa de Vulnerabilidade - SPV

Observando que o ambiente Groove foi concebido seguindo a arquitetura cliente/servidor onde alguns dos serviços oferecidos pelo ambiente são executadas do lado do servidor e outros do lado do cliente, o SPV se enquadra como um dos serviços do Groove que executa do lado do cliente, consultando a base de dados de testes e vulnerabilidades do Groove que fica armazenada no servidor.

O cliente, após ter feito seu cadastramento no Groove e seu cadastro ter sido liberado, pode escolher a opção consulta na *home-page* incial do ambiente, indicando que deseja solicitar uma pesquisa sobre as

vulnerabilidades existentes em seu sistema. Essa solicitação acionará o protocolo PVU, visando autenticar o usuário para usar o serviço, conforme descrito na seção anterior. Após essa autenticação, é então liberado para o usuário o serviço SPV, onde o usuário pode fornecer uma chave de pesquisa que pode ser, por exemplo: **SO + VSO**. Isso provocaria uma consulta à base de dados Groove retornando para o usuário uma lista com todas as vulnerabilidades cadastradas para aquela versão, daquele sistema operacional.

Observamos que esse processo é executado parte do lado do servidor e parte do lado do cliente, considerando que a aplicação é um *applet* e que é executada na memória do cliente. No entanto, toda a operação com as bases de dados SQL é feita do lado do servidor.

Para exemplificar o funcionamento do SPV, suponhamos que um determinado usuário, deseje saber quais as vulnerabilidades existentes em todas as versões do sistema operacional Linux. O usuário, após ser devidamente autenticado pelo PVU, verá as opções apresentadas na figura 6.5.

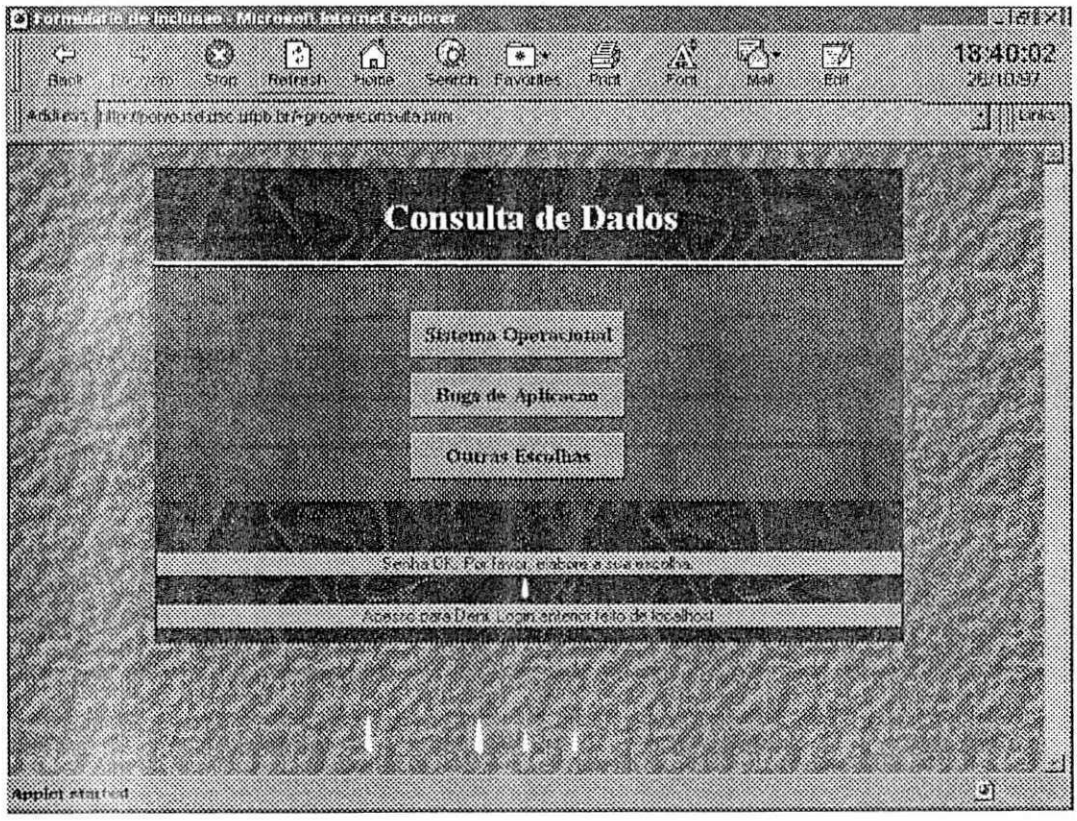


Figura 6.5: Home-Page de opções de consulta do Groove

Sabendo que em nosso exemplo o usuário necessita saber informações a respeito de todas as vulnerabilidades existentes em uma versão específica do sistema Linux, ele deve então escolher a opção **Sistema Operacional**, pois nessa opção, é apresentada ao usuário uma tela para a digitação do nome do sistema operacional a ser consultado e sua versão, como mostra a figura 6.6.

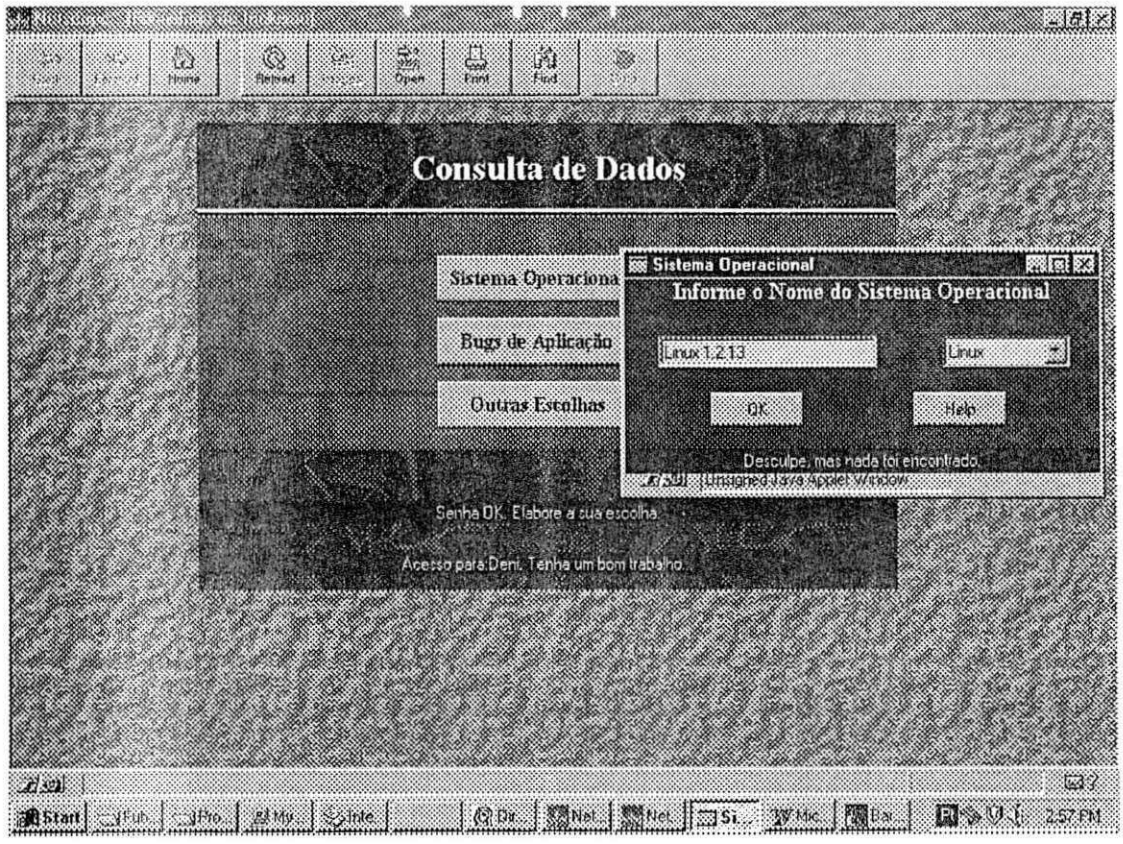


Figura 6.6: Tela para a entrada do nome do sistema operacional

Uma vez fornecido o nome do sistema operacional, o SPV realiza a consulta a base de testes e vulnerabilidades, retornando uma relação com todas as vulnerabilidades encontradas na versão 1.2.13 de sistemas Linux. A *interface* onde o resultado dessa pesquisa é apresentado é mostrado na figura 6.7.

Figura 6.7: Home-Page Groove mostrando o resultado da pesquisa sugerida na figura 6.6

O usuário pode também querer saber quais as versões vulneráveis de determinada aplicação - sendmail, por exemplo. Para isso, ele pode escolher a opção **Bugs de Aplicação** e então digitar, na tela mostrada na figura 6.10, o nome da aplicação.

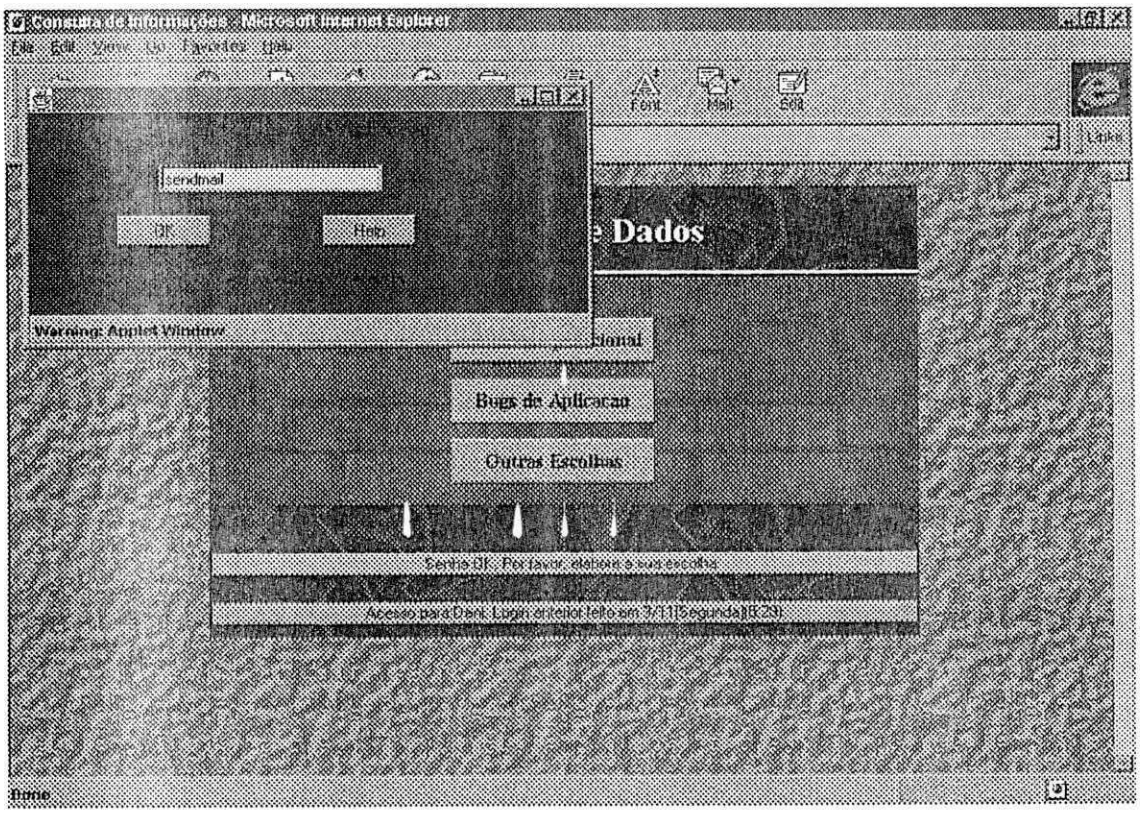


Figura 6.8 - Exemplo da interface da opção Bugs de Aplicação do Groove

Isso fará com que o SPV retorne com uma lista contendo as versões vulneráveis do sendmail. Um exemplo dssa lista é mostrado na figura 6.11.

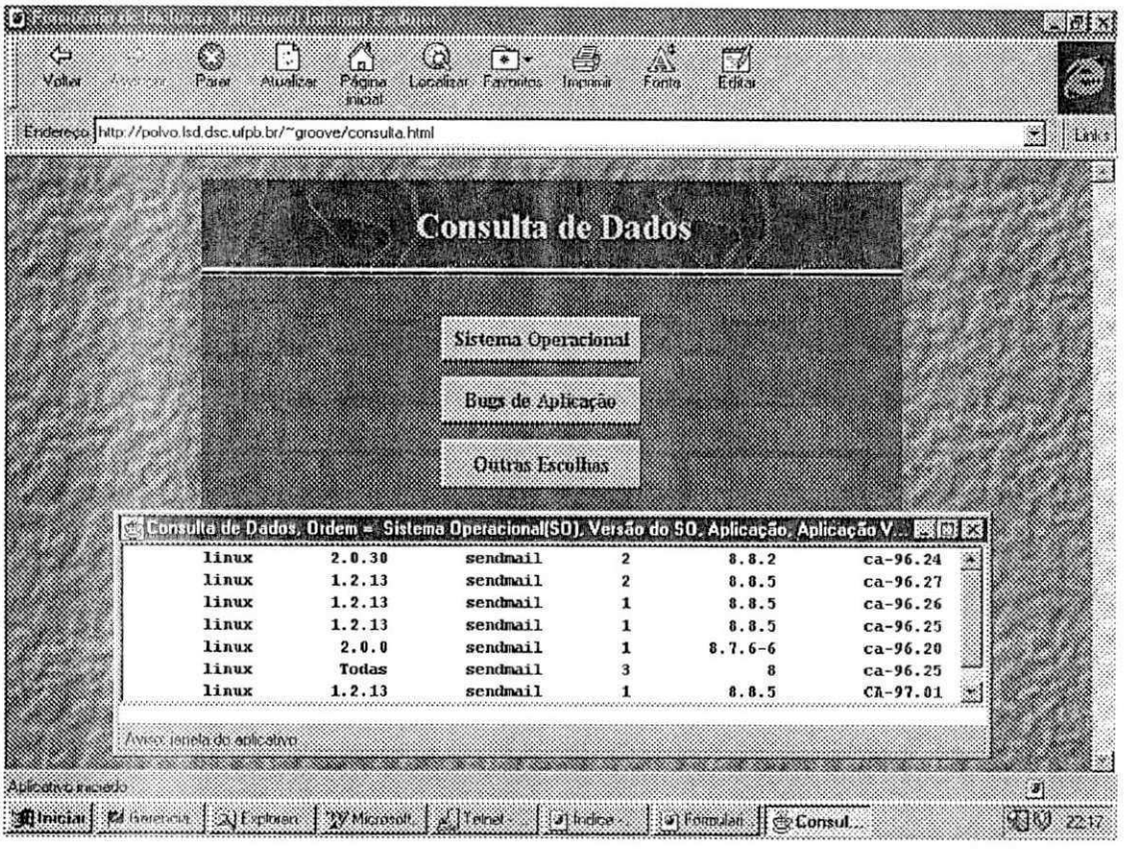


Figura 6.9 - Exemplo da interface de saída do Groove

Notamos ainda que o resultado de qualquer tipo de pesquisa mostra, além dos campos solicitados pelo usuário, um campo que aponta para uma página *web* contendo um texto explicativo sobre cada uma das vulnerabilidade listadas. Um exemplo disso é mostrado na figura abaixo.

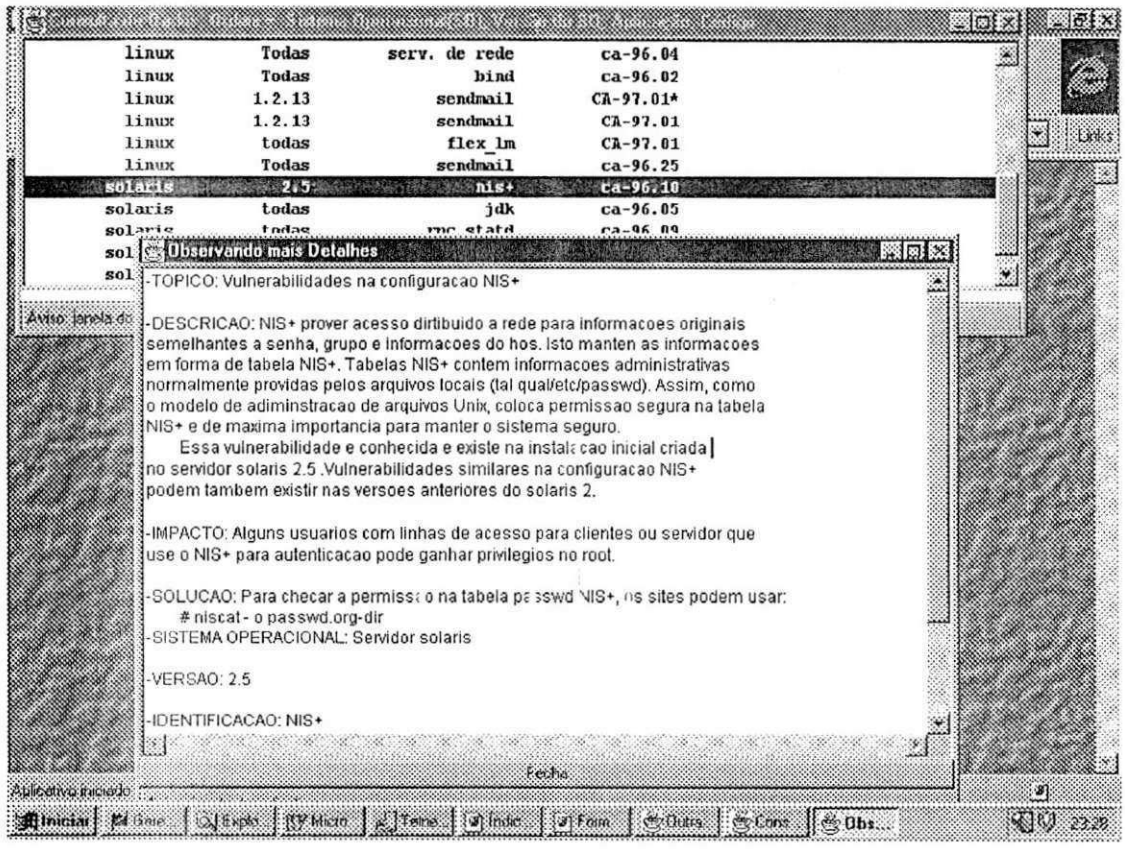


Figura 6.10 - Exemplo da interface de saída do Groove

Além disso, se o usuário desejar fazer uma outra consulta qualquer à Base de Dados Groove, ele pode então montar sua chave de pesquisa, escolhendo a opção outras escolhas. Essa opção dará ao usuário a opção de ver uma listagem geral de nossa base ordenada por algum dos campos que a compõe e a opção de combinar campos da base de dados visando gerar uma chave de pesquisa. Por exemplo, se o usuário quiser pesquisar por todas as versões da aplicação *sendmail* que possuem vulnerabilidade classificada como *buffer overflow*; essa pesquisa só será possível se combinarmos os campos **IDENT** e **SUBCLASS** (identificação da aplicação e sub-classe a qual ela pertence respectivamente). A figura abaixo ilustra nosso exemplo. Ainda observando a figura abaixo, notamos que o usuário pode montar sua chave de pesquisa fazendo qualquer combinação com os

campos existentes na base de dados. São considerados para formar a chave de pesquisa somente os campos preenchidos.

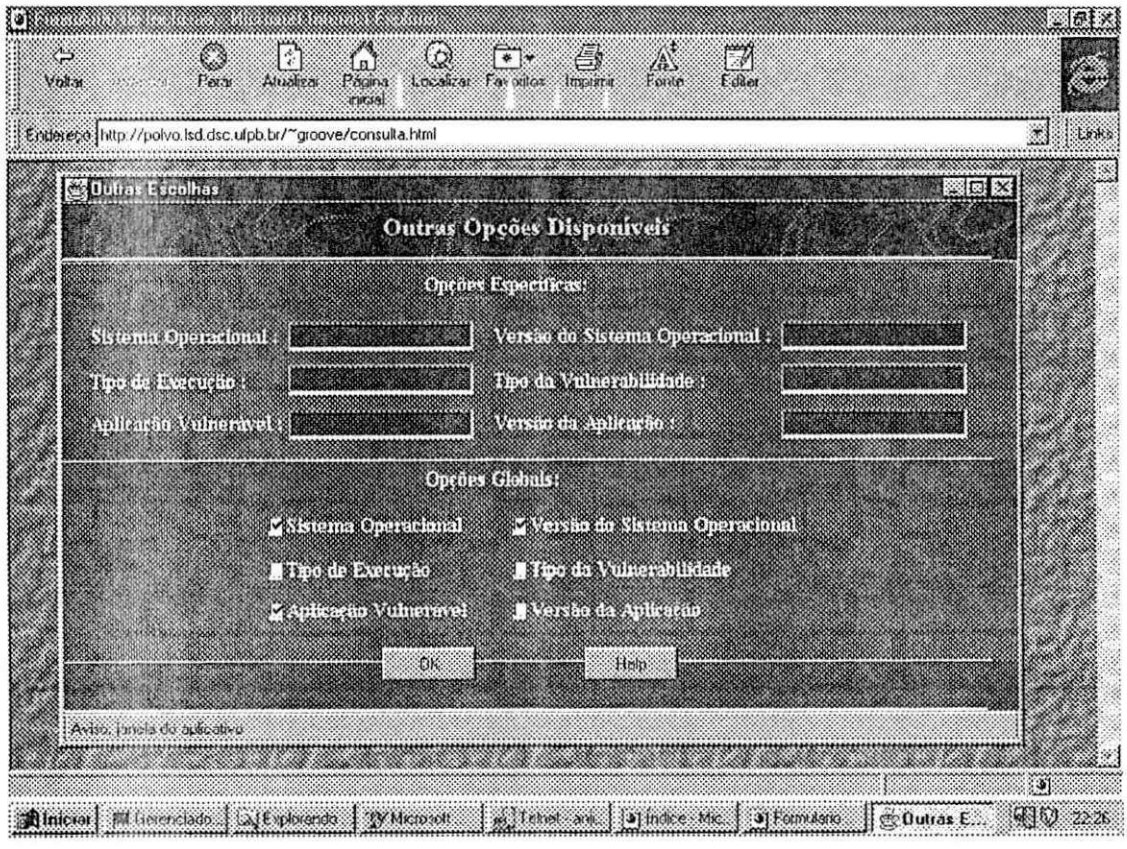


Figura 6.11 - Seleção da Consulta "Outras Opções"

Em nosso exemplo, o usuário deve marcar as opções Sistema Operacional, Versão do Sistema Operacional e Aplicação Vulnerável, como mostrado na figura 6.10. Isso fará com que o SPV consulte a base de dados de testes e vulnerabilidades, produzindo uma saída como a exemplificada na figura 6.12.

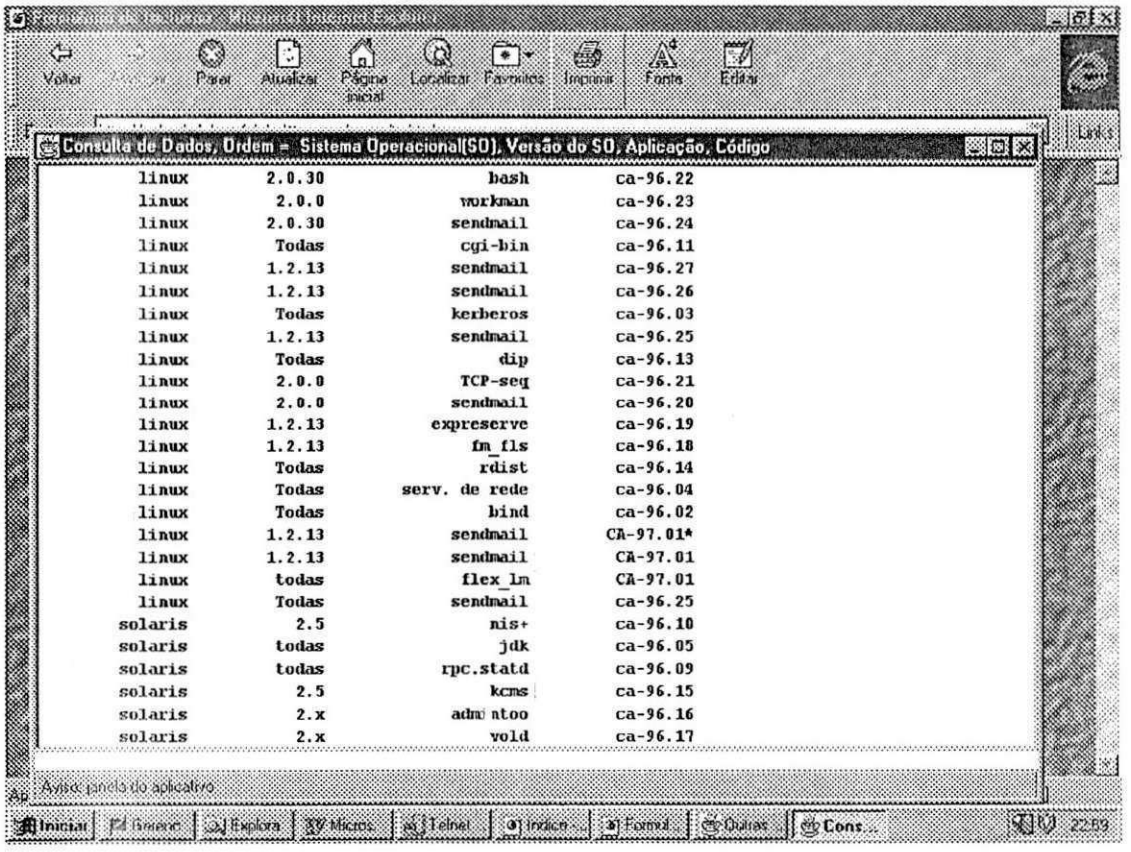


Figura 6.12 - Exemplo da interface de saída do Groove

Se o usuário preferir montar sua chave preenchendo os campos sistema operacional, versão do sistema operacional e aplicação vulnerável, por exemplo, ele então terá que preencher os seguintes campos mostrados na figura 6.13.

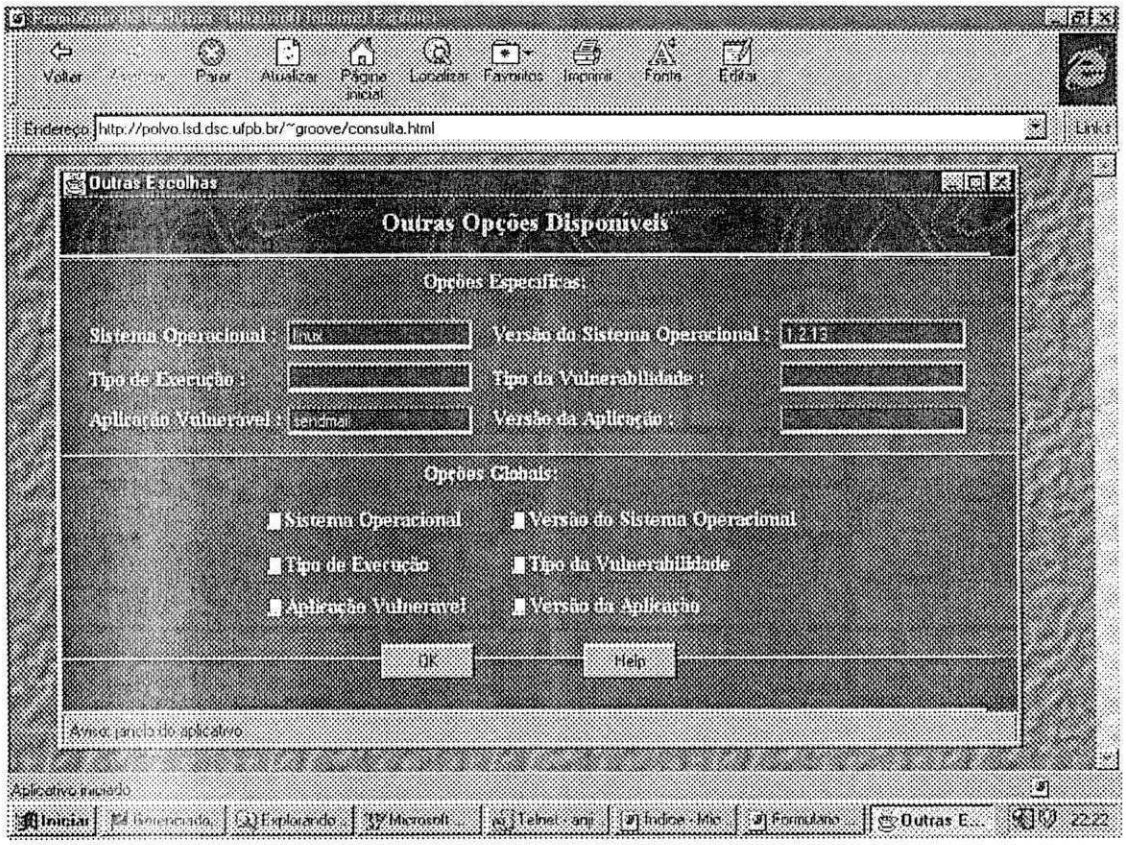


Figura 6.13 – Exemplo de Interface de saída do Groove

6.3.2. O Verificador de *Sites* Internet - VSI

O VSI é uma aplicação que é executada do lado do cliente, provocando acessos a arquivos localizados no servidor Groove. Esses arquivos são gerados pelo GBT toda vez que a base de dados de testes e vulnerabilidades for atualizada. Os arquivos devem ter o seguinte formato:

<Nome-do-Sistema-Operacional_teste.zip>: Contém todos os *scripts* de testes do Groove para o determinado sistema operacional. Por exemplo, o arquivo Linux_teste.zip deve conter todos os testes que devem ser aplicados em sistemas Linux.

<Nome-do-Sistema-Operacional_ex.zip>: Contém todos os *exploits* do Groove para o determinado sistema operacional. Por exemplo, o arquivo

Solaris_ex.zip deve conter todos os *exploits* que devem ser aplicados em um sistema operacional Solaris.

O VSI possui uma metodologia de testes definida em duas fases:

Fase A: Nesta fase, o VSI deve aplicar os *scripts* de testes no sistema alvo, visando encontrar versões de aplicações que estejam vulneráveis.

Fase B: Nesta fase, o VSI deve aplicar os *exploits* responsáveis por explorar uma determinada vulnerabilidade encontrada no sistema alvo.

A Fase B da metodologia visa aumentar o nível de confiabilidade dos testes, considerando que, durante a Fase A, os testes são feitos analisando apenas a versão da aplicação existente no sistema do usuário. Essa informação não garante que uma aplicação não tenha sido modificada por um invasor antes dos testes iniciarem. No Entanto, isso não invalida a Fase A, considerando que muitas vezes vulnerabilidades são divulgadas por fabricantes muito antes de atacantes terem desenvolvido os respectivos *exploits*. Sendo assim, os testes que verificam as versões das aplicações podem ser muito úteis, pois a informação da versão da aplicação vulnerável é, comumente divulgada junto com a própria vulnerabilidade.

Para acessar o VSI, o usuário deve escolher o botão *download* na *home-page* inicial do Groove. Uma vez solicitado o serviço de *dowlodad,* o usuário verifica duas opções: a de efetuar o *download* da aplicação VSI ou da aplicação Imaged. Optando pelo VSI, o usuário irá disparar um processo de ftp para a transferência do código da aplicação VSI para dentro do disco local de sua máquina. Feito isso, a instalação do VSI consiste simplesmente em descompactar o arquivo chamado vsi.tar. Durante a descompactação, todos os parâmetros necessários para a execução do VSI são selecionados.

Após executar o VSI, o usuário tem três opções: a de iniciar os testes, a de consultar as informações de contato e a de ajuda, como mostra a figura 6.14.

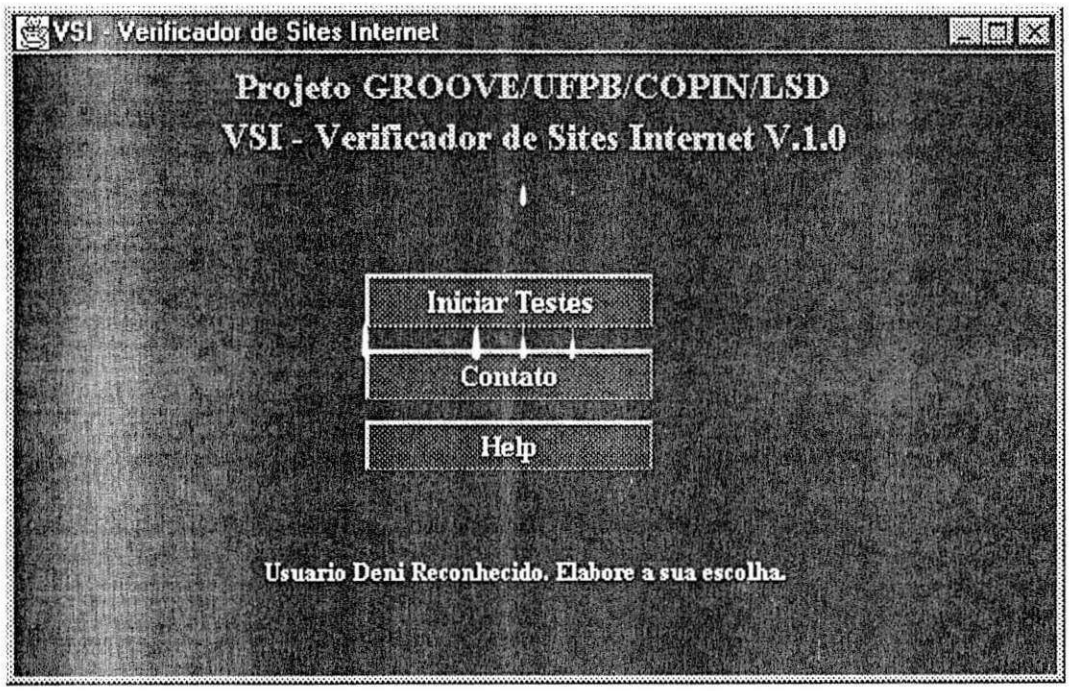


Figura 6.14 - Exemplo da interface inicial do VSI

Uma vez escolhida a opção iniciar testes, o VSI irá, antes de iniciar os testes, fazer uma chamada ao PVU visando autenticar o usuário, baseado em seu cadastro; essa verificação é feita pelo conjunto senha+endereço eletrônico, além de ser verificado se o usuário está executando o VSI a partir da mesma rede de onde ele se cadastrou. Essa verificação pode ser visualizada na figura 6.15.

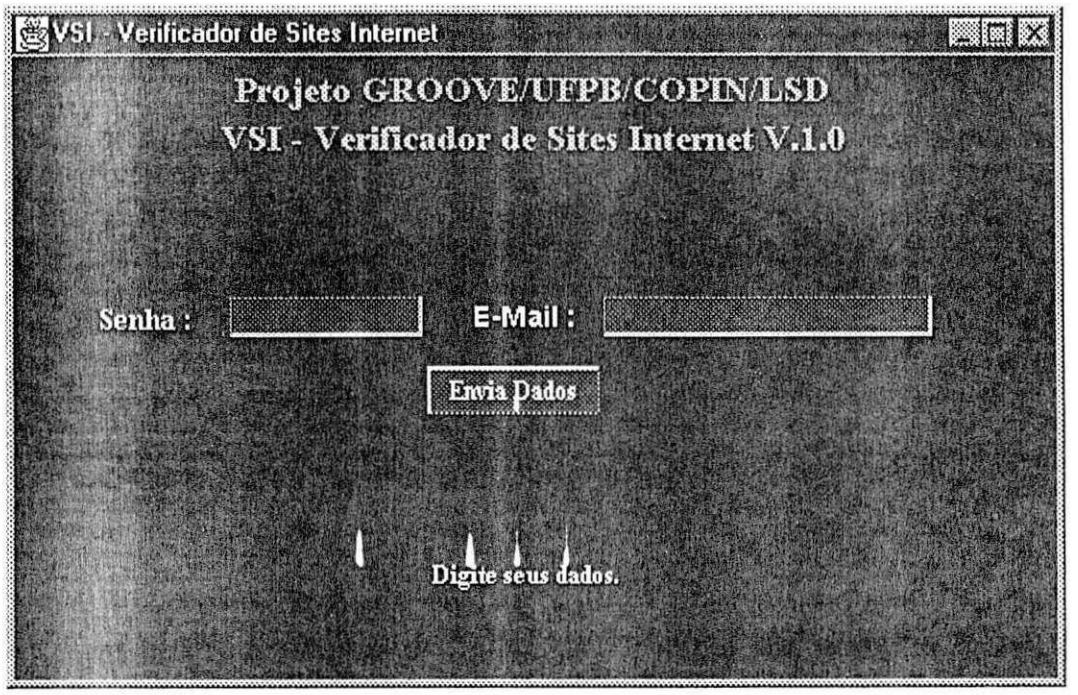


Figura 6.15 - Exemplo da interface do VSI chamando o PVU

Depois de validado o usuário, o VSI abre uma conexão ftp com o servidor Groove, visando transferir os arquivos que realizam as fases A e B dos testes, respectivamente. Esses arquivos são gravados no disco local do usuário e lá descompactados para então gerarem os arquivos de testes e de *exploits* que serão aplicados pelo VSI.

Após a execução dos testes, o VSI apresenta um relatório na tela do usuário contendo os resultados, informando quais as vulnerabilidades existentes em seu sistema, como mostra a figura 6.16.

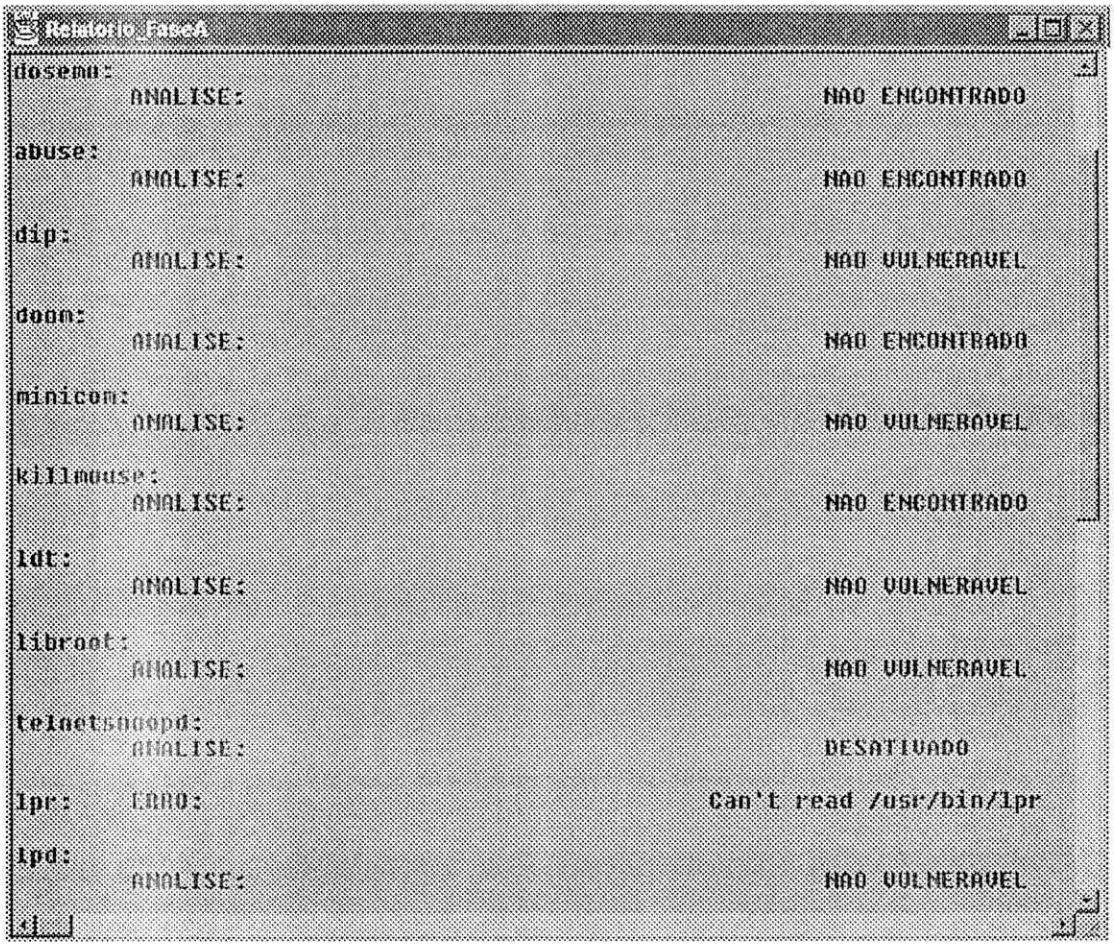


Figura 6.16 - Exemplo da interface do VSI exibindo o resultado dos testes.

ASPECTOS DE SEGURANÇA

Considerando o aspecto de segurança, resolvemos armazenar na base de dados de testes apontadores apenas para os binários das funções de *exploits*, os quais foram em sua totalidade construídos em linguagem C. O código de cada um desses *exploits* contém um parâmetro de linha de comando, que o usuário desconhece. Este parâmetro somente é conhecido pelo VSI, que então fica responsável pela sua execução.

Notamos que os binários dos *exploits* e os *scripts* de teste serão removidos do disco do cliente tão logo o VSI termine de aplicá-los.

No caso do usuário conseguir obter um desses binários de *exploits* antes do VSI removê-lo, o mesmo terá que conhecer o parâmetro de execução, que está criptografado dentro do código do *exploit,* para poder usá-lo. Isso minimiza a possibilidade do uso indevido desses programas *exploits*.

6.3.3. O Imaged

A aplicação *imaged* foi concebida após verificarmos que a grande maioria dos ataques a *sites* Internet resultam na criação/alteração/remoção de alguns arquivos importantes do sistema operacional. Normalmente esses arquivos são trocados por outros desenvolvidos pelo atacante do sistema. São os chamados *root/daemon kits*.

Considerando que o sistema operacional UNIX possui controle baseado em datas e tamanho de arquivos, resolvemos propôr um serviço na forma de um *daemon*, que seja capaz de realizar, durante sua instalação, uma "fotografia" de toda a área de sistema de um disco UNIX, gerando um arquivo criptografado com essa informação. Uma vez gerado esse arquivo, o *daemon* tem a função de fotografar o sistema periodicamente visando comparar o estado atual dos arquivos com o estado original armazenado durante a sua instalação. Isso permite ao administrador monitorar seu sistema em busca de diferenças em seus arquivos de *log* e de comandos, principais alvos dos ataques.

Para conseguir o Imaged, o usuário deve escolher, na home-page inicial do Groove, a opção download que apresentará a opção de transferir o código do Imaged para seu disco local. O usuário pode escolher qual a plataforma UNIX ele usa, visando fazer a transferência do binário correto para o seu sistema operacional. Após ser feita a transferência, a instalação do *Imaged* consiste de três passos básicos: descompactar o arquivo *imaged.tar*, configurar o arquivo *imaged.conf* e iniciar o binário *imaged*.

O *Imaged* possui um. arquivo de configuração, o *imaged.conf*, onde ficam armazenados todos os parâmetros para a sua execução.

O usuário pode configurar os seguintes parâmetros:

- a periodicidade da comparação;
- o conjunto de arquivos/diretórios a serem comparados;
- a maneira como os resultados da comparação são mostrados.

Um exemplo do conteúdo do arquivo de configuração **imaged.conf** é mostrado abaixo na figura 6.17.

```
### projeto groove
### universidade federal da paraíba
### image.conf → arquivo lido por imaged.
#### tempo
time                    5
##### diretórios
target /                -r
target /var             -r
target /usr             -r
##### exibição
output e-mail           root@dsc.ufpb.br
### fim
```

Figura 6.17: Exemplo de arquivo imaged.conf usado pelo imaged

Notamos que o usuário pode editar o arquivo *imaged.conf* e alterar os parâmetros de execução do *imaged.* O *imaged.conf* contém 3 seções a saber:

Seção *time*: especifica o tempo em minutos entre uma comparação e outra;

Seção *target*: Nessa seção existem alguns parâmetros a considerar: o primeiro parâmetro é o nome do arquivo ou do diretório que deve ser "fotografado"; se o primeiro parâmetro é um diretório, é possível ainda

especificar alguns parâmetros. O parâmetro "-r" indica que a imagem deve ser gerada de forma recursiva a partir do diretório especificado; além do parâmetro "-r", é possível especificar uma lista de arquivos/diretórios a serem excluídos da imagem; isso é feito através do parâmetro "-e". Se o usuário deseja que o *daemon* fotografe um diretório recursivamente, basta que ele coloque o parâmetro "-r" ao lado direito do nome do diretório. Por outro lado, se o usuário não quiser a recursividade, basta omitir o parâmetro "-r", caracterizando que deve ser considerado apenas o diretório especificado e seus arquivos. O usuário ainda pode optar por querer a recursividade exceto para alguns arquivos ou diretórios. Para isso basta que seja inserida como parâmetro no *imaged.conf*, uma lista com os nomes dos arquivos ou diretórios que não devem ser considerados.

Seção *output*: especifica onde serão mostrados os resultados da comparação. Nessa seção, podem ser passados os parâmetros ***mail***, ***console*** e ***file***. O parâmetro ***mail*** indica que o Imaged deve exibir os resultados de seu processamento via e-mail (nesse caso é necessário que seja informado o endereço eletrônico para onde deve ser enviado o e-mail; o parâmetro ***console*** indica que o imaged deve enviar seus resultados à console do sistema; e o parâmetro ***file*** indica que o imaged deve armazenar as saídas geradas em um arquivo (nesse caso deve ser informado o nome do arquivo).

Na figura 6.18, mostramos um exemplo do *imaged.conf*, preparado para:

❶ Executar de 5 em 5 minutos;

❷ Fotografar os diretórios */* recursivamente, exceto o ***/home***; o ***/var*** exceto o ***/var/spool/mail*** ; e o ***/usr*** sem recursividade;

❸ Exibir os resultados por e-mail para o *root* do sistema, para o arquivo *imaged.log* e para a console do sistema.

```
# projeto groove
# universidade federal da paraiba
# image.conf → arquivo usado pelo imaged.
# tempo
time            5
# diretórios
target /                -r  -e /home
target /var             -r  -e /var/spool/mail
target /usr
# exibição
output e-mail           root@dsc.ufpb.br
output console
output file             imaged.log
```

Figura 6.18: Arquivo imaged.conf

6.3.4. O Gerenciador da Base de Testes - GBT

Considerando que o Groove possui uma base de testes bem definida em sua estrutura, resolvemos propôr também o GBT, que implementa todas as funcionalidades de manipulação das bases de dados (teste e usuários).

Com o GBT, o administrador do Servidor Groove pode incluir, excluir e alterar dados contidos na base de testes. As funções do GBT são disponibilizadas apenas para usuários que pertençam a equipe de suporte do Servidor Groove, devidamente autorizados pelo PVU.

O GBT tem a função ainda de procurar periodicamente na base de dados de testes e vulnerabilidades, as referências aos arquivos de testes e de *exploits* adicionados à base, visando atualizar os arquivos utilizados pelo VSI durante a aplicação dos testes no sistema alvo. Ou seja, o GBT pesquisa a base de dados de testes e vulnerabilidades do Groove e seleciona todos os registros onde o campo FUN esteja preenchido, gerando um arquivo temporário. Após selecionar esses registros, ele faz uma pesquisa nesse arquivo temporário, separando os registros que possuem o campo TIPO (tipo do *exploit*: 0=script; 1=executável) preenchido com "0" em um arquivo e os que possuem o

campo TIPO preenchido com "1" em outro arquivo. Esses dois arquivos serão então compactados pelo GBT e depositados em um diretório de download do Servidor Groove para serem usados pelo VSI.

6.4. Detalhes de Implementação

Para a implementação das ferramentas do Groove, foram usadas as seguintes ferramentas:

a) mSQL - Mini SQL server versão 1.1.3 para ambiente Linux 2.0.0;

b) JDK - Java Development Kit versão 1.1.4;

c) API-mSQL-Java - Aplication Programming Interface mSQL-Java versão 1.0.2.

As ferramantas GBT, PVU e SPV foram todas implementadas em Java, utilizando *applets* e recursos da API *mSQL-Java*. A ferramenta VSI foi implementada como uma aplicação java *standalone*, necessitando que o usuário Groove possua em sua máquina uma máquina virtual java para interpretar seu código. O Imaged, por sua vez, foi implementado em linguagem C, possuindo hoje versões compiladas para várias implementações de BSD de UNIXes.

As ferramentas do Groove estão em sua primeira versão, por isso ainda apresentam problemas inerentes a isso. Por exemplo, o VSI somente pode ser executado em máquinas UNIX, pois os testes que estão armazenados na base de dados enfocam apenas sistemas desse tipo; o SPV possui características que modificam sua aparência em determinadas versões de navegadores *web*; e o Imaged só pode ser executado com as opções default, ou seja, ele ainda não processa o arquivo *imaged.conf*.

A implementação do Groove não parou e nem deve parar, estamos trabalhando para aperfeiçoar o ambiente no intuito de fornecer um serviço cada vez mais completo aos nossos usuários.

Capítulo 7
Conclusões

7.1. Groove vs. Outras Ferramentas

No ambiente Groove, existem 5 ferramentas, como já vimos no capítulo anterior. Comparando as funcionalidades entre cada uma das ferramentas mencionadas acima, chegamos a conclusão que as funcionalidades das ferramantas COPS, ISS e SATAN são semelhantes as funcionalidades do VSI - Verificador de *Sites* Internet e do SPV - Serviço de Pesquisa de Vulnerabilidades, pois verificam *sites* Internet, além de fornecer informações sobre vulnerabilidade. A vantagem das ferramentas do Groove, em nosso ponto de vista é a facilidade de instalação (o usuário não necessita compilar a aplicação para executá-la), diminuindo a necessidade de conhecimento para usar e o suporte que o servidor Groove tem, mantendo sempre atualizadas suas bases de dados.

Notamos que as ferramentas de verificação de segurança expostas no capítulo 1 desenvolvem funcionalidades específicas, enquanto que o Groove procurou agrupar 85% dessas funcionalidades em um ambiente integrado que permita ao usuário realizar sempre uma consulta atualizada sobre vulnerabilidades em sistemas, assim como realizar testes em seus sistemas com um nível a mais de confiabilidade, considerando que no VSI, usamos, além da estratégia tradicional de testes levando em consideração apenas versões de aplicações e serviços inicializados, a estratégia de aplicar *exploits* no sistema alvo, de forma segura mas que garanta que determinada vulnerabilidade exista ou não no sistema alvo.

Além disso, o Groove também oferece o *Imaged*, um *daemon* configurável pelo usuário e distribuído de forma pré-compilada. Com o *Imaged*, o administrador poderá ficar sabendo de qualquer alteração ocorrida nos arquivos de seu sistema, de forma a poder descobrir qualquer tentativa de ataque.

O ambiente Groove oferece também um nível de controle de segurança no intuito de saber "de onde" e "quem" está usando nossas ferramentas, visando controlar a aplicação nossos testes. Para isso o Groove conta com o PVU que autentica usuários baseado em seus cadastros. Essa autenticação é feita primeiro pela verificação de uma senha, seguido pela verificação do endereçco eletrônico do usuário para finalmente ser validado seu endereço IP. Com isso, sabemos que determinado usuário só poderá executar o VSI a partir de um determinado domínio.

7.2. Comentários Finais

Com a pesquisa desenvolvida, conseguimos alcançar os objetivos definidos, pois conseguimos compilar e gerar 350 vulnerabilidades existentes em *sites* Internet, assim como a forma de explorá-las e corrigi-las. Essas vulnerabilidades foram caracterizadas através de uma classificação que possibilitou a criação de uma base de dados de consulta que possibilita a um usuário obter informações organizadas em um único repositório de dados sobre os tipos de vulnerabilidades possivelmente presentes em seu *site* e uma explicação detalhada de como elas podem ser exploradas.

O trabalho trouxe informações importantes também aos componentes da pesquisa, considerando que a instituição onde desenvolvemos o trabalho também se caracteriza como um *site* Internet e, portanto, está sujeito a uma série de vulnerabilidades. As informações aqui obtidas valem também para os nossos próprios administradores aqui na UFPB. O VSI foi utilizado em nosso *site* e detectou várias falhas, contribuindo com o trabalho de implantação de segurança da equipe de suporte de nosso *site*. O SPV

também foi utilizado por estudantes e membros da equipe de segurança do Labcom, contribuindo para solucionar dúvidas sobre vulnerabilidades.

Além disso, deixamos informações disponíveis para que programadores de aplicações acadêmicas ou comerciais, possam consultar visando não deixar vulnerabilidades em seus códigos.

Um resumo dos objetivos alcançados por esse trabalho segue abaixo:

a) Organizar o conhecimento relativo às vulnerabilidades existentes nos sistemas operacionais que suportam *sites Internet* e nas aplicações comumente disponíveis nesses *sites*;

b) Mostrar como as principais aplicações de sistemas operacionais e do conjunto de protocolos TCP/IP estão vulneráveis, e dessa forma gerar subsídios que possibilitem os desenvolvedores de aplicações evitar ou pelo menos limitar vulnerabilidades em novas aplicações desenvolvidas;

c) Ajudar a reduzir a "ingenuidade" de alguns *sites Internet*, oferecendo uma ferramenta que detecta as vulnerabilidades mais comuns e sugerindo procedimentos para eliminá-las. Com isso, é possível contribuirmos para o aumento do nível de segurança de alguns *sites*, além de oferecermos um conhecimento organizado aos administradores desses sistemas sobre os procedimentos básicos de teste para manter mais seguras suas máquinas e informações.

7.3. Trabalhos Futuros

Para trabalhos futuros, sugerimos um estudo para a elaboração de uma metodologia para a verificação/interceptação de intrusos e ataques ao sistema, utilizando o modelo de classificação produzido pelo ambiente

Groove, ou seja elaborar métodos que tentem, além de verificar se um sistema está (ou foi) atacado, interceptar o ataque e rastreá-lo. Além disso, sugerimos que seja consolidado o ambiente Groove através da construção de ferramentas que façam mais do que verificar o nível de segurança de um sistema, ou seja, ferramentas que possam realmente **garantir** níveis de segurança em *sites* Internet.

Sugerimos também um estudo que vise produzir uma forma viável de atualização automática da base de dados de testes e vulnerabilidades do Groove, visando eliminar o trabalho manual de atualização que hoje temos. Acreditamos que com conceitos de agentes inteligentes e o estabelecimento de padões que levem em consideração alguns dos campos de nossa base de dados, seja possível a criação de um Agente Atualizador Inteligente.

Referências Bibliográficas

[1] Cerf, V. and E. Cain. ***The DOD Internet Architecture Model***. Computer Networks, 1983.

[2] Cerf, V. and R. Kahn. ***A Protocol for Packet Network Interconnection***. IEEE Transactions of Comunications, 1983.

[3] Comer, D. E. and J. T. Korb. ***CSNET Protocol Software: The IP-to-X25 Interface***. Computer Communications Review, 1985.

[4] Comer, D. E. ***Operating System Design - The XINU Approach***. Prentice-Hall, 1984.

[5] Comer, D. E. ***Operating System Design Vol II - The XINU Approach***. Prentice-Hall, 1984.

[6] Comer, D. E. and D. L. Stevens. ***Internetworking with TCP/IP Vol I - Architecture e Protocols***. Prentice-Hall, 1994.

[7] Comer, D. E. and D. L. Stevens. ***Internetworking with TCP/IP Vol II - Design, Implementation and Internals***. Prentice-Hall, 1993.

[8] Comer, D. E. and D. L. Stevens. ***Internetworking with TCP/IP Vol III - Client-Server Programming and Applications, BSD socket version***. Prentice-Hall, 1993.

[9] Comer, D. E. and D. L. Stevens. ***Internetworking with TCP/IP Vol III - Client-Server Programming and Applications, AT&T TLI version***. Prentice-Hall, 1993.

[10] Denning P. J. The Science od Computing: ***The ARPANET After Twenty Years***. American Scientist, 1989.

[11] Ritchie, D. M. and K. Thompson. ***The UNIX Time-Sharing System***. Communications of ACM, 1978.

[12] Curry, David A . ***UNIX System Security: A guide for users and Systems Administrators. Reading***, MA: Addison Wesley, 1992.

[13] Farrow, Rik. ***UNIX System Security***. Reading, MA: Addison Wesley, 1991.

[14] Ferbrache, David e Gavin Shearer. ***Unix Instalation, Security e Integrity***. Englewood Cliffs, NJ: Prentice Hall, 1993.

[15] Grampp, F T e R H Morris. ***UNIX Operating System Security***, AT&T Bell Laboratories Technical Journal, October 1994.

[16] Reid, Brian. ***Reflections on Some Recent Widespread Computer Break-ins***. Comunications of the ACM, Volume 30, Number 2, February 1987.

[17] Wood, Patrick H. e Stephen G. Kochan. ***UNIX System Security***. Camel, IN: Hayden Books, 1986.

[18] D. Anderson et al. ***Next Generation Detection Expert System (NIDES).*** Software Design, Product Specification, and Version Description Document, Project 3131, July 1994.

[19] S.M. Bellovin. ***There Be Dragons***. Proc. Third USENIX UNIX Security Symp, pp 1-16, Baltimore, Sept, 1992.

[20] M. Bishop. ***A Taxonomy of UNIX System and Network Vulnerabilities.*** Technical Report CSE-95-10, Univ. of California at Davis, Sept, 1995.

[21] P. Brinch Hansen. ***Reproducible Testing of Monitors.*** Software Pratice and Experience, vol 8, pp 721-729, 1978.

[22] D.E. Denning. ***An Intrusion-Detection Model.*** IEEE Trans. Software Eng. Vol. 13, no. 2, pp 222-232, Feb 1987.

[23] C. Dowell and P. Ramstedt. ***The COMPUTERWATCH Data Reduction Tool.*** Proc. 13th Nat'l Ccmputer Security Conf. pp. 99-108, Washington, D.C., Oct, 1990.

[24] D. Farmer and W. Venema. ***Improving the Security of Your Site by Breaking into It.*** USENET posting, Dec. 1993.

[25] D. Farmer and E.H.Spafford. ***The COPS Security Checker System.*** Proc. Summer USENIX Conf. pp. 165-170, June 1990.

[26] J. Hochberg et al. NADIR: ***An Automated System for Detecting Network Intrusion and Misuse***. Computers and Security, vol 12, no. 3, pp. 235-248, May 1993.

[27] S.E. Smaha. Haystack: ***An Intrusion Detection System***. Proc. IEEE Fourth Aerospace Computer Security Application Conf. Orlando, Fla, Dec. 1988.

[28] E.J. Weyuker and B. Jeng. ***Analyzing Partition Testing Strategies***. IEEE Trans. Software Eng. Vol. 17, no. 7. Pp. 703-711, July 1991.

[29] K. Zhang. ***A Metodology for Testing Intrusion Detection Systems***. MS thesis, Univ. of California at Davis, May 1993.

[30] R. Lippmann and H.M. Heggestad. ***Lincoln Laboratory Intrusion Detection Reserch.*** Proc. 4th Computer Misuse and Anomaly Detection Workshop. National Security Agency. Ft. Meade, Maryland, to appear.