UNIVERSIDADE ESTADUAL DE CAMPINAS

INSTITUTO DE COMPUTAÇÃO

DEPARTAMENTO DE SISTEMAS DE COMPUTAÇÃO

Relatório Parcial de Iniciação Científica

Plataforma de Teste para Segurança de Redes

Bolsista
Daniel Pupim Kano

Orientador
Prof. Dr. Paulo Lício de Geus

10 de junho de 2001

Plataforma de Teste para Segurança de Redes

Bolsista:
Daniel Pupim Kano
Instituto de Computação
Universidade Estadual de Campinas
daniel.kano@ic.unicamp.br

Orientador:
Prof. Dr. Paulo Lício de Geus
Instituto de Computação
Universidade Estadual de Campinas
paulo@ic.unicamp.br

2. INTRODUÇÃO

A Internet é hoje uma integradora de sistemas, empresas, negócios e pessoas, tornando possível a comunicação em um tempo antes impossível de ser realizada por outros meios, e assim entregando um novo ritmo mais dinâmico a esta era em que vivemos. Esta rede de computadores está baseada no conjunto de protocolos TCP/IP que é o foco central deste projeto.

Num primeiro momento foi feito um estudo dos protocolos em questão junto com a utilização de ferramentas existentes que auxiliam o entendimento dos conceitos e a visualização destes em um sistema real. Em seguida foi feita a confecção de uma biblioteca de manuseio de pacotes para poder realizar uma mímica da atuação da pilha TCP/IP para poder servir de experiência para a elaboração da plataforma em seguida.

Com os conceitos aprendidos e reproduzidos tem-se uma nova tarefa: a concepção da plataforma e sua implementação. Foi decidido implementar uma linguagem em se possa construir e configurar uma máquina de estados e para isso se fez necessário aprender a utilização de um analisador léxico e um construtor de "parseador" para a realização da tarefa. E assim, com a experiência de programação e o estudo inicial, dá-se início á criação da linguagem e sua gramática e os procedimentos a serem realizados dada cada uma de suas declarações.

E é neste ponto em que se encontra o projeto, na elaboração da linguagem, sua gramática e ações.

3. Produção

3.1 Protocolos e pacotes

Os pacotes de dados em uma rede são as partículas atômicas que contém informação. O acesso a estas partículas provêem informações intrínsecas ao tráfego da rede, tanto a própria informação sendo transitada quanto informações específicas das máquinas em cada uma das pontas desta rede são expostas. De posse destes dados e com a participação ativa neste fluxo de pacotes, é possível ter o controle de todo o ambiente de comunicação para então poder realizar ações que prejudicam a confiança e o boa convivência na rede.

O conjunto de protocolos TCP/IP é organizado em camadas sendo que cada uma delas é responsável por uma parte da comunicação. Essencialmente são quatro camadas: uma camada é responsável pela gerência do hardware de rede e com sua interface com o sistema operacional (camada de enlace); uma segunda camada tem a tarefa de gerenciar o movimento dos pacotes dentro da rede, em outras palavras, o roteamento dos pacotes entre duas máquina (camada de rede); uma outra camada fica responsável pelo fluxo de dados entre dois *hosts* (camada de transporte); e finalmente uma última camada fica responsável pelos detalhes particulares de cada aplicação (camada de aplicação).

3.1.1 Biblioteca produzida

Paralelamente ao estudo feito, foi se implementando uma biblioteca de funções (para uso interno e didático) que lidasse com pacotes de redes TCP/IP, obtendo o seguinte resultado:

- “las_fill_checksum.c”: Funções padrão de cálculo de checksum.
- “las_fill_ip_header.c”: Função de preenchimento de cabeçalho IP.
- “las_fill_tcp_header.c”: Função de preenchimento de cabeçalho TCP.

- “las_pcap.c”: Funções para a manipulação de uma outra biblioteca, a libPcap, responsável pela obtenção de pacotes provenientes da rede e a entrega destes ao programa mediante a critérios de filtragem.
- “las_tcp_opts.c”: Contém uma função que insere ao cabeçalho TCP suas opções que fazem com que o tamanho do cabeçalho seja variável e está de acordo com RFC793. Esta área de opções do protocolo TCP é muito relevante para análise de segurança redes pois a forma como as opções estão dispostas no pacote reflete as diferenças de implementação da pilha TCP/IP de diferentes sistemas operacionais. Por exemplo, ao enviar um pacote, cujo protocolo da camada de transporte é o TCP, com o *flag* SYN ativo e a seguinte seqüência de opções: <Window Scale> <No operation>
  <Maximum Segment Size> <Timestamp> <Echoed MSS> (WNMTE para encurtar).
  A implementação da pilha TCP/IP do Windows NT4/95/98 retornaria um pacote com o *flag* ACK ativo e com a seqüência de opções, MNWNNT enquanto que a implementação do Linux 2.2.12 retornaria uma pacote com o *flag* ACK ativo e a seqüência de opções, MENW.
  A estas peculiaridades de implementação de cada sistema operacional dá-se o nome de OS-Fingerprint (Impressão digital do sistema operacional). Uma poderosa ferramenta para obtenção de informação remota de computadores via uma rede TCP/IP , o NMAP[1], tem contribuído em muito neste projeto.

Para confeccionar e para capturar pacotes foram elaboradas (simples) bibliotecas, mas para envio dos pacotes, devido à facilidade de se “abrir” e lidar com um socket em modo “raw” (cru), ou seja, um socket de comunicação que tenha acesso à rede e que deixe que o processo que o está utilizando enviar pacotes cujos cabeçalhos foram preenchidos pelo próprio processo, não se fez uma função que cuidasse do envio do pacote.

Com estas funções foi possível reproduzir algumas funções realizadas pela pilha TCP/IP tais como:

[1] www.insecure.org/nmap

- Estabelecer conexão com um servidor TCP (3-way-hadshake) enviando dados e terminando a conexão.
- TELNET "sniffing": Uma simples técnica de observar o tráfego de rede (estando todas as máquina da rede conectadas por um hub), capturar todos os pacotes com porta TCP 23 (TELNET), tratar o simples protocolo de comunicação e controle do serviço TELNET e mostrar no monitor tudo o que estiver na área de dados do cabeçalho TCP e que não for de controle de interno do serviço, faz com que se consiga visualizar toda a comunicação (de maneira integral, inclusive a senha e o login do cliente) entre o cliente e o servidor. Isto mostra a fragilidade e a insegurança de se utilizar o serviço de TELNET, pois todos os dados estão trafegando abertamente na rede e basta haver uma máquina intermediária mal intencionada para que toda a privacidade do usuário do serviço deixe existir, e por isso que se recomenda a utilização de comunicação criptografada, tal como o serviço de SSH.
- "*Spoofing*" local: Possuindo acesso ao cabo de rede[1], por onde passam todos os pacotes provenientes de todas as transações de dados, pôde-se facilmente estabelecer uma conexão TCP com uma outra máquina conectada neste mesmo meio físico falsificando a própria identidade, o que mostra a falta de autenticação no conjunto básico de protocolos TCP/IP.
- Detetor de "*sniffing*": Quando a interface de uma máquina está em modo promíscuo, a pilha TCP/IP desta máquina passa a receber todos os pacotes provenientes da rede, não verificando o endereço MAC (considerando uma rede Ethernet) dos pacotes. Uma maneira de detectar se alguma máquina está em modo promíscuo é alterar o endereço MAC da máquina suspeita na tabela ARP da máquina que vai efetuar tal verificação e enviar um pacote que provoque uma resposta desta máquina, por exemplo um "ping". Se a máquina suspeita estiver em modo promíscuo a pilha TCP/IP dela vai receber este pacote, pois não houve verificação de endereço MAC (que está errado), e irá responder à máquina que enviou o pacote, o que não aconteceria se ela não estivesse em modo promíscuo.

3.1.2 Ferramentas utilizadas

As principais ferramentas utilizadas neste projeto, até o momento, foram aquelas que possuíam a capacidade de absorver pacotes provenientes do meio físico da rede para poder visualizar o tráfego e que proporcionasse um meio de interação com a rede e estas foram:

- TCPDUMP: ferramenta que captura de pacotes que passam pela interface de rede e mostra de maneira organizada os dados colhidos, embora com uma interface não muito amigável. O TCPDUMP é o núcleo de muitas outras ferramentas de captura de pacotes e é um dos primeiros programas do gênero. No Linux, o TCPDUMP utiliza o BPF (Berkeley Packet Filter) modificado para filtragem.
- ETHEREAL: ferramenta de atuação idêntica à anterior, mas com uma interface gráfica muito organizada, além de mostrar não apenas os cabeçalhos dos pacotes de maneira interpretada, mas também interpreta vários serviços do nível de aplicação. Ou seja, enquanto o TCPDUMP mostra apenas os cabeçalhos organizadamente e os dados da camada de aplicação em um bloco hexadecimal, o ETHEREAL é sensível também ao nível de aplicação.
- NMAP: Uma das mais completas ferramentas para *scanning* criada nos últimos anos, pois possui diversas técnicas para tal, e consegue obter informações importantes sobre máquinas remotas que podem servir tanto para o desenvolvimento da segurança quanto para o próprio ataque da segurança.

3.2 Plataforma

Esta plataforma, que é o resultado final deste projeto, foi concebida com o objetivo de prover ao usuário facilidades de montar pacotes possibilitando preencher os cabeçalhos e inserir dados nele de acordo com seus conhecimentos dos protocolos. E com os pacotes construídos, que são as partículas atômicas deste projeto, dá-se início ao gerenciamento da ação destes em uma rede de computadores, pois ao usuário será possível a configuração do

[1] Possuir uma interface de rede em modo promíscuo, de modo a obter todos os pacotes provenientes da rede.

comportamento da plataforma, sendo a partir de uma dada a inserção de pacotes no tráfego de rede, ações serão executadas de acordo com as respostas recebidas, podendo ser o envio de novos pacotes uma ação, ou seja, a configuração desta plataforma será o estabelecimento e a configuração de estados e suas ações, assim como a definição das entradas que levam a eles.

3.3 Pacotes

No ambiente UNIX, por ter como linguagem padrão o C, mais especificamente no Linux, existem arquivos de cabeçalho (header files - *.h) que definem, entre outras coisas, as estruturas de dados que modelam a forma dos cabeçalhos dos protocolos de comunicação de acordo com os padrões estabelecidos nas RFCs, por exemplo, os arquivos contidos nos diretório /usr/include/netinet, neles está contida, de maneira bem fechada, a estrutura dos pacotes.

Mas, com o objetivo de possuir uma certa a flexibilidade estas estruturas pré-definidas não são utilizadas, e assim o usuário poderá entregar uma nova interpretação aos campos dos pacotes de acordo com as mudanças que vão ocorrendo com os protocolos ao longo do tempo, como por exemplo, o campo Type Of Service do cabeçalho do protocolo IP que, hoje, pode ter a interpretação de Differentiated Services (RFC2475).

Assim, achou-se melhor a criação de arquivos onde são definidas as estruturas de dados dos pacotes, não de maneira rígida como as estruturas em C (struct), mas sim um arquivo representando o cabeçalho de cada protocolo onde cada linha é um campo do cabeçalho que possui como informações o tamanho em bytes (ou em bits), o offset (a posição relativa ao início do cabeçalho) e o nome do campo, por exemplo, a definição do cabeçalho IP padrão(sem opções) ficaria assim:

Nome	Tamanho	Offset	Comentário
IPVersion	4b	0	Versão do protocolo
IPHdrlen	4b	4b	Tamanho do cabeçalho do pacote
TOS	1B	1B	Type of service
Totlen	2B	2B	Tamanho total do pacote
ID	2B	4B	Identificação
Frag_flags	3b	6B	*Flags* de fragmentação

Frag_off	13b	51b	Offset de fragmentação
TTL	1B	8B	Time to live
Proto	1B	9B	Protocolo da camada seguinte
Cksum	2B	10B	Checksum
Src_addr	4B	12B	Endereço origem
Dst_addr	4B	16B	Endereço destino

*(b = bits B = bytes)

Os dados dispostos assim flexibilizam a filtragem de pacotes como será visto logo a seguir.

3.4 Filtragem

Nesta plataforma deverá haver o controle de dois tipos de filtros, um filtro que restringe a entrada de pacotes, provenientes da rede, ao kernel que são de interesse exclusivo da plataforma, e outro que traga os pacotes que chegam ao dispositivo de rede para a plataforma de acordo com as necessidades da mesma.

Percebe-se que ambos os filtros são intrínsecos aos sistemas operacionais em que irão ser utilizados e portanto decidiu-se enfocar o ambiente UNIX, neste caso, o Linux. Para o primeiro filtro o Linux provê módulos de filtragem de pacotes que pode ser configurado de acordo com as necessidades requeridas, tais como IPChains (kernel 2.2.x) ou IPTables (kernel 2.4.x). Para o segundo filtro existe a implementação do BPF(Berkeley Packet Filter) para o linux que, através de uma biblioteca já existente, o Libpcap, provê tais facilidades de captura de pacote no nível de enlace de acordo com regras de filtragem estabelecidas, uma vez que não se deseja obter todos os pacotes recebidos pelo dispositivo de rede, mas sim os interessantes à plataforma.

Tendo capturado os pacotes, uma segunda filtragem é feita para se fazer uma análise destes para que então a plataforma decida para qual novo estado partir, pois os pacotes recebidos são um dos agentes responsáveis pela mudança de estado desta máquina de estados.

Esta segunda filtragem é mais atômica e para torná-la flexível suficiente a abstração de campos (entende-se por campos, campos de uma *struct* em C) não mais existe (neste

projeto), e agora se trata diretamente com os bytes (ou bits) e para isso será feito uso dos arquivos de definição dos cabeçlhos dos protocolos mencionados na seção 3.3, por exemplo, se o usuário desejar comparar o campo Dst_addr de um pacote, será feita uma tradução e a comparação será feita com os 4 bytes da 16ª posição (contada em bytes) a partir do início do cabeçalho IP do pacote recebido.

Percebe-se aqui que, aparentemente, esta pequena mudança de procedimento em nada mudou efetivamente, pois quando se tratava com campos de uma *struct* na verdade também era isto que acontecia, mas do ponto de vista da implementação esta foi uma radical mudança, pois esta generalização torna configurável a interpretação empregada nos cabeçalho dos pacotes, o que antes não era muito fácil de ser feito, senão quase impossível, e assim, um ponto positivo para a flexibilidade que se deseja alcançar.

3.5 Estrutura de Dados

Cada um dos estados são referenciados pelos seus nomes (string de caracteres) e por isso está sendo utilizado uma tabela de *hashing* que aponta para estuturas de dados dos estados e que tem como chave a soma dos valores ASCII dos caracteres do nome dos estados, inicialmente um simples algoritmo de *hashing* está sendo usado, com o tamanho da tabela de *hashing* três vezes maior que o número total de estados, o que implica que o usuário deverá saber de antemão o número total de estados que irá utilizar em sua máquina.

A estrutura de dados dos estados está definida como uma *struct* em C com os seguintes campos:

- STATE_ID: número de identificação do estado
- STATE_NAME: string que indica o nome do estado
- ACTION: valor referente a uma ação
- ARG: argumentos referentes a ação escolhida
- NEXT: State_id do próximo estado

O valor "*action*" indica a tarefa do estado em questão, que pode ser:

- send_pack: Envia um pacote

- recv_pack: Recebe um pacote e muda de estado de acordo com o pacote recebido
- wait: Espera um dado tempo antes de ingressar no próximo estado

Em *arg* irá conter os parâmetros que devem ser passados para cada uma das ações dos estados, por examplo, no caso de send_pack, o valor de arg será o número de identificação do pacote a ser mandado.

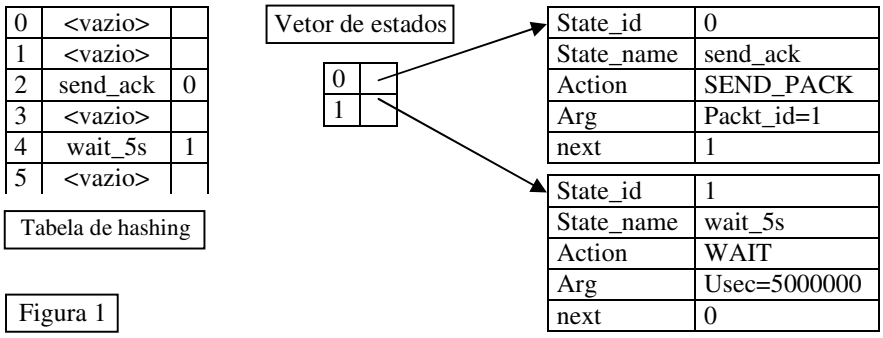

Figura 1

No exemplo da figura 1 acima pode-se verificar uma simples máquina de estados representada de acordo com a estrutura de dados descrita acima, esta máquina possui apenas dois estados, “send_ack” e ”wait_5s”, sendo o primeiro responsável pelo envio de um pacote cujo número de identificação é 1, e o outro responsável por um tempo de 5 segundos de atraso (acima indicado em microsegundos 5000000us), e assim esta máquina de estados fica indeterminadamente enviando pacotes em um intervalo de 5 segundos.

Com a máquina de estados montada em memória, numa segunda etapa um interpretador irá percorrer esta estrutura de dados realizando cada uma das tarefas de cada etapa.

3.6 Linguagem

Foi decidido que uma boa maneira de se implementar esta máquina de estados configurável é criando uma linguagem que permita esta interação do usuário com a plataforma e assim foi necessário o estudo de como torná-la possível. Em Lex&Yacc[1]

[1]J. Levine, T. Mason e D. Brown. Lex & Yacc. O’Reilly, 1992

informações sobre o analisador léxico LEX e o construtor de *parseador* YACC foram extraídas para a criação da linguagem.

3.6.1 Lex e Yacc

Estes dois programas ajudam a construir outros programas que transformam uma entrada em forma de texto estruturado em comandos para alguma tarefa desejada. Neste texto estruturado são aplicados duas ações: Primeiro se divide o texto, que até então não passa de um monte de *strings*, em pedaços de texto (chamados *tokens*) com sentido, a partir de expressões regulares pré-definidas. Ao LEX fica estabelecida esta ação de procurar tais expressões regulares no texto para então efetuar tal divisão.

Com o texto dividido em *tokens* faz-se necessário achar uma relação entre eles que defina expressões, declarações, blocos e procedimentos para se efetuar as tarefas desejadas, e para isso deve haver regras que definem tais relações que se dá o nome de gramática. O YACC obtém uma descrição concisa desta gramática e produz rotinas em C que farão parte do programa responsável por achar tais relações, o parseador.

3.6.2 Gramática

Concluiu-se que a gramática a ser definida deveria possuir declarações que fizessem tais ações:

- Iniciar a máquina de estados, ou seja, sua estrutura de dados.
- Criar estados e definir suas ações.
- Interligar os estados.
- Visualizar a máquina de estados.
- Configurar a filtragem feita pelo Kernel.

Até o momento a gramática e suas respectivas implementações se encontram assim:

- "**Init [number] states**": Inicia a máquina de estados com [number] estados, ou seja, cria a tabela de *hashing* de tamanho 3 vezes [number] e um vetor de tamanho [number] de apontadores para as estruturas de dados dos estados.
- "**create state [str]**": Cria as estruturas de dados referentes ao novo estado de nome [str], insere [str] na tabela de *hashing* e aloca memória para o próximo apontador livre do vetor de estados para ser preenchido com as informações deste novo estado.
- "**state [str] : snd_pack [number]**": Define como ação do estado de nome [str] o envio do pacote de número [number].
- "**state [str] : recv_pack**": Define como ação do estado de nome [str] o recebimento de pacotes (esta declaração não foi definida ainda).
- "**state [str] : wait [time]**": Define como ação do estado de nome [str] um atraso de [time]ms para partir para o próximo estado.
- "**krnl_setup**": Define as configurações de filtragem do kernel (esta declaração não foi definida ainda).
- "**show all**": Mostra todo o vetor de estados.
- "**show used**": Mostra todos os estados do vetor de estados.
- "**show state [number]**": Mostra o estado de número [number].
- "**show state [str]**": Mostra o estado [str].

3.7 Status

Primeiro Ano	Jan	Fev	Mar	Abr	Mai	Jun	Jul	Ago	Set	Out	Nov	Dez
Estudo inicial	o	o										
Filtros de captura de pacotes e de acesso	o	o	o									
Estados e temporização				o	o	o	-	-	-	-	-	-
Gerador de pacotes										-	-	-

o: Parte do cronograma executado.

-: Parte restante do cronograma.

4 Próximos passos

Pontos restantes para a conclusão da plataforma (versão 1º ano):

- Linguagem: Restam ser implementadas as declarações da gramática que relativas ao recebimento de pacotes, que são responsáveis pela mudança de estados, e à configuração da filtragem do *kernel*.
- Com o item acima concluído, tem-se toda uma estrutura de dados montada em memória relativa à máquina de estados desejada, e portanto uma seqüência lógica do projeto é a aplicação do conhecimento obtido, e das funções construídas, na primeira etapa do projeto e criar uma "entidade" que percorra esta estrutura da máquina de estados e que efetue cada um de seus passos.
- E, paralelamente a toda segunda parte do projeto, a estrutura de dados, a manipulação e armazenamento de pacotes (assim como definida em 3.3), será construída para melhor se adaptar aos moldes da estrutura da plataforma de maneira a ser bem flexível à alterações e inclusões de protocolos.

5 Bilbiografia

Stevens, W. R.. TCP/IP Illustrated, Volume I: The Protocols. Addison-Wesley, 1994.

Levine, J., Mason, T. e Brown, D.. Lex & Yacc. O'Reilly, 1992.

Stevens, W. R.. UNIX Network Programming, second edition Networking APIs: Sockets and XTI. Prentice Hall, 1998.

Comer, D. E., Stevens, D. L.. Internetworking with TCP/IP, Volume III: BSD socket version. Prentice Hall, 1993.

Arkin, O.. ICMP Usage in Scanning, 2000, http://www.sys-security.com.

Fyodor. The Art of Port Scanning e Remote OS Detection via TCP/IP Fingerprinting, 1997 e 1999, http://www.insecure.org.

Request for Coments

Linux Documentation Project, *Howtos e Mini-howtos, http://www.linuxdoc.org*