Ataque, defesa e contra-ataque: Invasão

Capítulo 20

"Abordar navios mercantes
invadir, pilhar, tomar o que é nosso(...)
Preparar a nossa invasão
E fazer justiça com as proprias mãos."[1]
RPM, "Rádio Pirata"

1. Do álbum Revoluções por minuto, de 1985.

"Por que o cachorro entrou na igreja"? Ante essa pergunta capiciosa, a maioria das pessoas, depois de "raciocinar" um pouco, disparam respostas das mais estapafúrdias e incongruentes que o ser humano pode ser capaz de conceber. "Poque o padre chamou". "Porque os anjos o conduziram para a plenitude espiritual". "Porque ouviu um chamado de Alá para destruir o templo de idolatria dos infiéis". "Porque quis". "Porque sim". "Porque havia quermesse e ele sentiu cheiro de lingüiça". "Porque estava no colo da dona". "Ah, vai te catá, mano"! Mas a resposta correta é a mais simples e lógica de todas. O cachorro entrou na igreja porque a porta estava aberta.

A partir deste ponto ultrapassamos os limites. A legislação brasileira não reconhece o acesso a sistemas que estejam abertos à Internet como invasão, da mesma forma como entrar num shopping center também não é invasão: as portas estão abertas para tal.

Pela lei norte-americana, entretanto, esse tipo de invasão (de sistemas e de shopping centers) já é considerado invasão de propriedade. Se você não é bem vindo em um sistema, pode ser processado se entrar. Se não é bem vindo em um shopping center, também pode! Houve pelo menos um caso, divulgado pela imprensa, de um cidadão americano processado por invasão de propriedade por estar na área pública de um shopping. Explico: ele estava com uma camiseta pacifista (a frase exata era "Give peace a chance", de John Lennon) e a segurança do shopping solicitou que ele retirasse a camiseta ou deixasse o prédio. Como recusou-se a ambas as coisas, foi preso e processado. Home of the free...

Cuidado, portanto, quando invadir sistemas geograficamente instalados lá ou pertencentes a empresas daquele país. Você será processado pelas leis de lá, e é provável que seja extraditado. Hackerismo = Terrorismo = Prisão Perpétua... Já deu pra notar, né?

A invasão em etapas

Assim como dividimos nosso ataque em seis passos diferentes, o passo quatro – invasão – pode ser dividido em duas etapas.

A primeira etapa é o **acesso a um host da rede**. Como vimos nos capítulos de redes, as redes internas das empresas normalmente não utilizam números IP roteáveis pela Internet. Portanto, temos que invadir primeiramente um computador limítrofe que possua duas interfaces de rede e, portanto, atenda a ambas. Normalmente é o próprio firewall, ou um proxy, mas é possível que falhas de configuração ou mesmo descuido possam escancarar outros sistemas.

Para a primeira etapa, um número limitado de técnicas pode se empregado. Entre eles, a procura por modems aceitando conexões externas (war dialing) e a exploração de falhas específicas através de exploits. Observe que nenhum sistema em especial é visado. Os crackers podem apontar suas miras tanto em servidores como em estações de trabalho e mesmo em componentes de rack como roteadores e afins!

A segunda etapa é mais trabalhosa que difícil. Uma vez conseguido o acesso à rede interna, passa-se à **invasão sistemática dos computadores dentro da rede**. Essa etapa pressupõe quebra de senhas e acesso a áreas restritas mesmo para quem está dentro. Dizemos que é trabalhosa porque, uma vez dentro, cada um dos sistemas autônomos disponíveis para os usuários internos requer toda aquela metodologia que vimos: Planejamento, Observação, Busca, Invasão, Manutenção e Evasão. E não esqueça: vários microataques 1-2-1-3-1-4-1-5-1-6-1... Sempre corrija e ajuste seu planos! As coisas mudam, os dados mudam, o administrador muda e você sempre tem informações novas. Se quer sucesso, organize-se!

E, se você for um lammer, não tem o que fazer por aqui, já que você não quer saber de aprender nada. Vá brincar com os exploits publicados e deixe os profissionais trabalharem!

War Dialing + Brute Force

Redes bem configuradas permitem acesso a elas apenas por pontos bem específicos e controladíssimos, normalmente uma única conexão com a Internet. Empresas muito grandes possuem vários links, mas todos eles são (ou deveriam ser) controlados e monitorados com rigor. Entretanto, alguns funcionários "espertos" tentam driblar a lentidão ou os controles de acesso a sites não-autorizados com conexões discadas ou ADSL ligados diretamente a suas estações. Esses funcionários não têm noção do perigo real representado por essa prática: tais conexões são totalmente desprotegidas, e uma vez com acesso a essas conexões, o hacker já está dentro da rede, e pode queimar diversas etapas de sua invasão.

Em outras ocasiões, a própria empresa precisa proporcionar acesso discado a seus funcionários viajantes, ou aos que trabalham em casa. Para tanto, possuem uma bateria de modems e dezenas de números de telefone disponíveis para receber chamadas e conexões. Um prato cheio para hackers!

Sim, sim, não ia deixar de comentar que o termo War Dialing foi cunhado no filme War Games de 1983 em que Matthew Broderick discava para todos os números que ele conseguia atrás de modems livres. Mas isso provavelmente você já sabe. Se não, vá até sua locadora e peça o filme **Jogos de Guerra**. Se não é um filme excelente, pelo menos as referências tecnológicas estão corretas – dentro, é claro, dos limites da precisão hollywoodiana. Mais informações podem ser encontradas em orbita.starmedia.com/~necrose/Sci-Fi/Filmes/wargames.htm, www.movieprop.com/tvandmovie/reviews/wargames.htm e jbonline.terra.com.br/jb/papel/cadernos/internet/2001/08/15/jorinf20010815003.html.

Conheça suas armas

War dialing é guerra. Se você é um soldado, vai à guerra e não leva sua espada, está frito! As ferramentas usadas pelos war dialers são chamadas, normalmente de... war dialers. Uma busca no Google traria diversos sites com essas

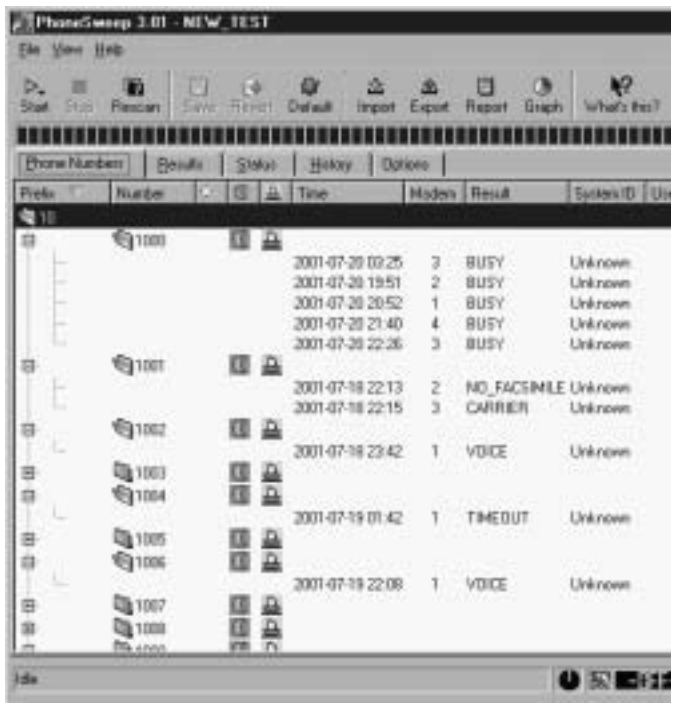
ferramentas, a maioria gratuitas e muitas delas com código fonte aberto e livre. Mas, em uma excepcional colher de chá, aí vai um link com algumas delas: neworder.box.sk/box.php3?gfx=neworder&prj=neworder&key=wardil&txt=Wardialers.

Recomendamos epecialmente, entre eles, o THC Scan (www.thc.org). É a ferramenta n° 1 de War Dialing hoje em dia. Foi desenvolvida para DOS, mas pode ser usada em Unix com emuladores, como o Dosemu.

Se você quer dar uma olhada em como as coisas começaram, procure por Demon Dialer e pelo ToneLoc. São ferramentas parecidas com as que os Phreakers usavam na época do filme de Broderick, incluindo o próprio mock-up da cena.

Para Unix, além do excelente THC Scan + Dosemu há alguns nativos como o WARD, cuja descrição está no endereço www.securiteam.com/tools/6T0001P5QM.html e pode ser baixado e www.0xdeadbeef.info/code/ward.c; ou pérolas como o Jericho e o ShockDial, ambos encontráveis em www.securityfocus.com/tools/category/26.

Há também muitos war dialers comerciais, destinados a permitir que empresas testem suas instalações à procura de modems perdidos. Uma das mais completas é o Phone Sweep (www.sandstorm.net). Além da excelência do software, a empresa promete consultoria e suporte por assinatura (com pagamento mensal).



Uma lista de equipamentos reconhecidamente vulneráveis e detectáveis pelo Phone Sweep podem ser encontrados em *www.sandstorm.net/products/phonesweep//sysids.shtml*.

Outra opção é o TeleSweep, da Securelogic. Mais informações em telesweepsecure.securelogix.com.

Nossa recomendação: baixe, instale e brinque com, pelo menos, as ferramentas gratuitas listadas aqui. Você vai descobrir que, nesta época de ADSL e Cable Modem, ainda existem MUITOS modems discados recebendo chamadas por aí, especialmente em empresas.

Outra dica: você pode usar os discadores dos provedores de acesso como War Dialers improvisados!!! A maioria deles possuem um arquivo externo com a lista de números de telefone de conexão. Basta substituir essa lista por uma especialmente criada, e você terá um war dialer instantâneo, inclusive com rediscagem automática e varredura de todos os números da lista. Baixe os discadores de todos os provedores que você se lembrar e verifique. Esta dica é muito útil quando se está usando um computador laranja como discador e não se quer levantar suspeitas com programas especializados. Um THC Scan levantaria suspeitas no computador do seu tio, mas o discador do iG (ou do Terra, do iBest, do UOL ou da AOL...) passaria despercebido, ainda mais se o seu tio for assinante de um desses provedores.

Ah, a força bruta é tão linda...

Certo, você descobriu números de telefone que possuem modems atendendo a chamados externos. Mas e agora, o quê fazer com eles? Entra agora em cena outro tipo de war dialer que, ao invés de varrer números atrás de modems, faz inúmeras tentativas de login em um mesmo número.

Se em outros tipos de controle de acesso por login e senha é possível encontrar outros meios de entrada além do brute-force, com modems a coisa é diferente. Você não tem qualquer outra informação além do prompt do sistema pedindo um usuário válido. Sua única saída é usar um programa de força bruta para tentar entrar. Logicamente, usar suas listas de palavras (que vimos nos capítulos sobre vulnerabilidades) e saber o nome de algum usuário ajuda muito aqui. Atenção lammers! Usar SQL injection em modems é a demonstração de burrice mais enfadonha que você poderia cometer. Não que faça algum mal, mas mostra claramente que você não entende nada de nada, mesmo... Vá brincar de carrinho ou de boneca!

Se você usou o THC Scan, um bom companheiro para ele é o THC Login Hacker. Baixe-o e veja como é fácil entrar, quando se encontra um modem "compreensivo". No site oficial há diversos programas, exploits e brute-forcers para muitos protocolos, incluindo SMB, HTTP, FTP e Proxy. Verifique em www.thc.org/releases.php.

A propósito, THC é acrônimo para The Hackers Choice... Apesar da arrogân-

cia e presunção do título auto-outorgado, as ferramentas são realmente muito boas. No site ainda encontramos diversos whitepapers sobre invasão e segurança dignos de leitura.

Entendendo o estouro de pilha (buffer overflow)

Vários exploits valem-se dos chamados buffer overflows para conseguir um shell nos sistemas vulneráveis a eles. Os script kiddies aplicam pesadamente tais exploits em suas presas sem saber exatamente o que são e como funcionam. Como queremos ter mais do que noções, na verdade, um entendimento completo de como essas ferramentas operam, teremos que escovar um pouco mais de bits, dessa vez olhando atentamente o código dos aplicativos. Não é preciso dizer que será necessário desenferrujar seus conhecimentos de linguagens de programação, mas nada de outro planeta.

Como o nome é estouro de pilha, obviamente há uma pilha a ser estourada (dãää!). Para entender como o estouro funciona, devemos, primeiramente, conceituar o que seria essa tal de pilha.

Um amontoado de coisas...

A palavra portuguesa pilha nos traz à cabeça duas imagens:

1. Uma pilha de pedras, de pratos, de moedas, de corpos no filme do Stallone...

2. Uma célula de energia elétrica usada em lanternas portáteis e brinquedos.

Peguemos a primeira imagem. Uma pilha de moedas, por exemplo, daquelas que o Tio Patinhas tem em cima de sua escrivaninha na caixa-forte. O adorável pão-duro de Patópolis arruma diligentemente suas moedinhas, uma sobre a outra. Com isso, quando vai guardá-las na caixa-forte o bom ávaro retira cuidadosamente a primeira moeda e a põe no saco de dinheiro, depois a segunda, depois a terceira... Observe um detalhe: a primeira moeda a ser retirada foi a ultima a ser colocada.

No exemplo da pilha de pedras, Fred Flintstone usando seu tratorossauro recolhe pedras retiradas da encosta e as empilha no local indicado pelo Sr. Pedregulho Slate, seu chefe. Note que, nesse caso, as pedras que Fred pega primeiro são as que vão ficar por baixo da pilha. Mas talvez o melhor exemplo (e mais útil) seja mesmo o da pilha de pratos. Podemos inclusive usar um exemplo de pilhas recursivas. Quando os pratos estão sujos na mesa e a louça do dia é sua (...), você recolhe-os e os empilha nas mãos ou em uma bandeja. Observe: o último prato que você pegou fica no topo da pilha. Quando chega à cozinha, faz o caminho inverso. Como não dá pra colocar a pilha toda de uma vez, você pega prato por prato e os empilha novamente, desta vez na pia. À medida que os vai lavando, você os empilha pela terceira vez (agora já limpos) do outro lado da cuba, e novamente o que era último ficou primeiro. Para guardá-los no armário... é, acho que você já entendeu.

E as pilhas de lanterna? Elas não têm muito haver com o papo agora. Só por curiosidade, o nome "pilha" é dado a esse elemento gerador de energia porque as primeiras células quimio-elétricas eram, er, bem, diversos discos de metais diferentes (parecidos com moedas, às vezes moedas mesmo!) alternados entre si e mergulhados em uma solução ácida ou salgada. Pensando bem, até tem haver...

As pilhas no reino digital

E o que têm os computadores com isso? Pegando emprestado a noção de pilhas, os programas em geral (e os sistemas operacionais em particular) podem pegar fragmentos de dados e os guardar em áreas de memória chamadas pilhas ou stacks. É uma maneira fácil de armazenar pequenos dados, pois num acessso convencional de memória o programa é obrigado a:

1. Definir a posição de memória onde o dado será guardado;
2. Definir o tamanho em bytes que o dado terá;
3. Reservar esse tamanho em bytes na posição definida;
4. Mandar o dado para essa posição;
5. Bloquear a posição para que não seja sobreescrita.

Para recuperar o dado, o programa terá que:

1. Lembrar a posição de memória onde o dado está;
2. Apontar para aquela posição;
3. Lembrar o tamanho dos dados em bytes;
4. Puxar esses bytes daquela posição de memória;
5. Liberar a posição para uso futuro.

No caso da pilha, não há necessidade de nada disso. O programa pode simplesmente:

1. Mandar dados para a pilha (push).

Na hora de recuperá-los, basta:

1. Puxar dados da pilha (pop).

É claro, como é um acesso seqüencial e não aleatório, caso queiramos usar o segundo dado da pilha antes teremos que retirar o primeiro dado. Isso traz dificuldades adicionais para o programa, que tem que "dar um jeito" de gerenciar os dados da pilha corretamente. Mas há aplicações onde essa abordagem é a melhor. Por exemplo, imagine que queiramos fazer um programinha que faça uma soma de três valores. A forma mais simples desse programa seria:

1. Chama função soma(a,b);

2. Obtém primeiro número e o entrega à função soma;
3. Obtém o segundo número e o entrega à função soma;
4. Coloca o resultado na pilha;
5. Chama função soma(a,b);
6. Obtém terceiro número e o entrega à função soma;
7. "Puxa" o topo da pilha e entrega à função soma;
8. Soma(a,b) faz a operação e joga o resultado na saída.

Observe que o resultado da primeira soma ficou armazenado na pilha, esperando ser chamado de volta ao fluxo normal do programa. Uma solução como essa envolvendo posições de memória demandaria funções para reservar memória para três variáveis, envio e recuperação triplicada de dados e possivelmente uma função soma mais complexa.

Esse exemplo foi didático, mas longe do ideal. Vamos ver um pequeno programa em C. Deve rodar de forma idêntica em qualquer plataforma, pois não usa bibliotecas específicas.

```
void funcao_idiota (void)
     {
            char xuxu[5];
            gets (xuxu);
            printf("%s\n", xuxu );
     }

main()
     {
            funcao_idiota();
            return 0;
     }
```

Não se preocupe, você não precisará ter visto um programa C alguma vez na vida para entender este aqui. Veja só: qualquer linguagem estruturada que se preze permite que você crie, com os comandos básicos que ela possui, funções mais complexas para serem usadas pelo programa principal. Em nosso exemplo (e em qualquer código C), o programa principal é "marcado" pela função main( ). Dentro das chaves { e } temos o programa completo, composto por duas outras funções:

```
main()
     {
            funcao_idiota();
            return 0;
}
```


funcao_idiota( ) chama uma função criada por nós mesmos, e declarada no início do programa. Já a próxima linha, return 0, indica que o programa deve encerrar nessa linha e retornar para o shell que a chamou.

Temos que declarar funcao_idiota( ) *antes dela poder ser usada. Então, vamos a ela!*

```
void funcao_idiota (void)
     {
            char xuxu[5];
            gets (xuxu);
            printf("%s\n", xuxu );
     }
```

A função é, a grosso modo, uma sub-rotina que pode ser usada várias vezes dentro do programa. Em nosso main( ) a usamos apenas uma vez, mas seria possível, se quiséssemos, usá-la em diversos locais do código. O que essa idiotice faz é: 1) criar uma variável chamada xuxu com um tamanho de 5 bytes; 2) usar a função gets( ) da linguagem C para pegar caracteres do teclado e jogá-los na variável xuxu; 3) usar a função printf( ) para jogar o resultado na tela.

Se você está em uma máquina Unix, pode compilar o programinha e testá-lo. Na maioria dos sitemas, use o comando:

```
$ cc -o idiota idiota.c
```

Sendo idiota.c o arquivo texto contendo o código do programa e idiota o arquivo executável gerado pelo compilador cc. No Linux e no FreeBSD use o gcc em vez do cc.

Para rodar o programa, digite:

```
$ ./idiota
```

Aparentemente, nada acontecerá. Tente digitar um caractere qualquer e pressionar a tecla <Enter>. O caracter será replicado na tela. É só isso o que o programinha faz.

Em uma máquina DOS ou Windows, o procedimento é parecido. Procure por algum compilador de linha de comando. Se não souber onde procurar, comece com alguns destes. Experimente TODOS e escolha seu preferido!

Para DOS podemos indicar:

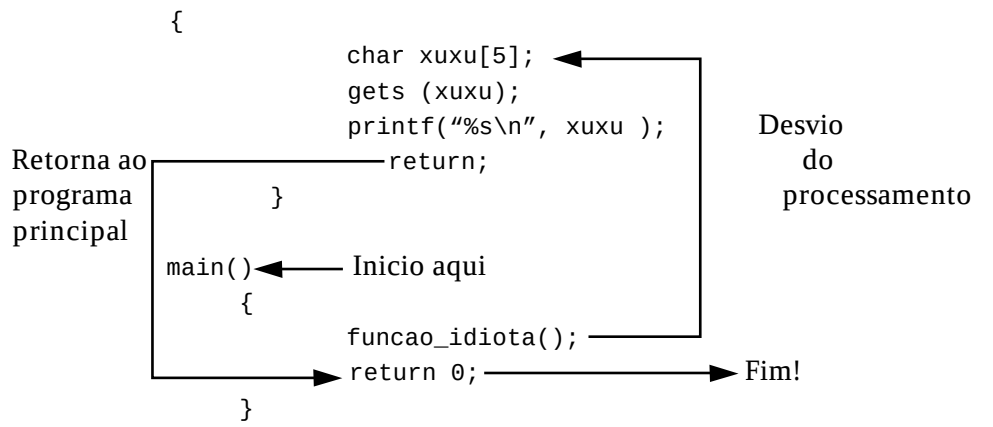
- DJGPP, o GCC do DOS: www.delorie.com/djgpp;
- O venerável Borland Turbo C 2.01 (Autorizado! Não é pirataria!): community.borland.com/museum ;
- Pacific C: www.elrincondelc.com/compila/pacific.html ;

Para Windows, existem, entre muitas, as seguintes opções:

- LCC Win32: www.cs.virginia.edu/~lcc-win32/index.html;
- DevCPP: www.bloodshed.net/devcpp.html (a escolha dos autores!);

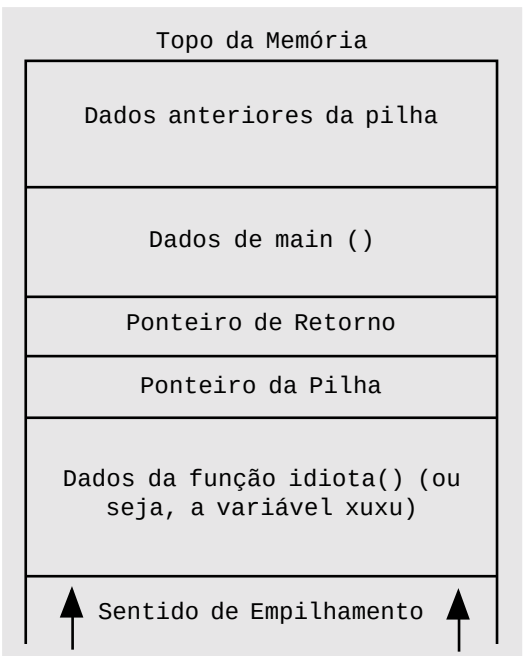
Aliás, em www.bloodshed.net existem links para compiladores de diversas linguagens! Confira!

Voltemos ao nosso programa. Quando o main( ) chama funcao_idiota( ), o processamento do programa principal é interrompido e desviado para a função. Quando a função termina seu processamento, retorna à rotina principal. Observe o código novamente, desta vez com o fluxo de processamento indicado por setas:



Bonito, não? Em princípio, funciona. Mas, para interromper o processamento de main( ), é neceesário colocar tudo o que main( ) esteja fazendo em algum lugar, desviar o processamento para funcao_idiota( ), processar o que estiver lá e depois retorar a main( ).Além de devolver os possíveis valores da função chamada (o que não é nosso caso – para simplificar não passamos argumentos entre as funções), também precisamos ter um meio de saber ONDE na memória estava o processamento de main( ) para que retomemos o processo.

Complicado? Também achei. Uma ilustração ajuda bastante!



Nooofaaaa (Com F mesmo. Lingua presa...) ! Mas não era uma pilha? Porque está de cabeça para baixo? Bem, normalmente as pilhas são armazenadas de cima para baixo, sendo o dado mais antigo na posição mais alta de memória e a pilha crescendo em direção à posição mais baixa. Pense nela como uma pilha de moedas feita no teto em direção ao chão. Talvez você tenha que usar cola para prender as moedas, mas ainda assim é uma pilha.

Como dissemos, os dados de main( ) são jogados na pilha. Note que ela não precisa estar necessariamente vazia, podendo conter, por exemplo, dados do shell ou da janela onde o programa foi chamado. Depois do main( ), também é guardado no stack um ponteiro, chamado de ponteiro de endereço de retorno ou return address pointer. É ele que indica ao processamento onde encontrar a próxima instrução depois do desvio (onde é que eu estava mesmo?). Em nosso caso específico, o return pointer guarda o endereço de memória onde reside a instrução return 0.

Depois do ponteiro de retorno, o sistema coloca um ponteiro de pilha, que aponta para uma tabela com dados de controle sobre a própria pilha – que obviamente o leitor compreende ser necessária. Por último, vão os dados temporários de nossa função secundária, chamada por main( ). Em nosso caso, é a variável xuxu, criada pela nossa função funcao_idiota( ).

Quando a função secundária termina seu processamento, os dados dela são retirados da pilha, depois o ponteiro de controle da pilha, depois o ponteiro do endereço de retorno. Quando o processamento é retomado na posição original, os dados de main são puxados da pilha e tudo volta a ser como era antes. Será?

Debug is on the table[2]

Lembremos de dois detalhes vistos nos parágrafos anteriores. O segundo tenho certeza que foi assimilado pelo leitor apenas como curiosidade, o primeiro deve ter passado despercebido:

1. A variável xuxu foi declarada como contendo apenas cinco bytes;
2. A pilha armazena dados de baixo para cima.

Lá atrás nós testamos nosso programinha assim:

```
$ ./idiota
a
a
$
```

Quando o processamento chegou na função gets( ), digitamos a letra "a". A função gets( ) colocou "a" dentro da variável xuxu, que foi impressa na linha seguinte pelo printf( ). Lembre-se de que xuxu tem um tamanho de apenas cinco bytes. O que acontece se passarmos de cinco?

2. Aula de inglês geek? Essa frase realmente foi dita, durante uma conversa entre eu e o diretor de informática de uma empresa de aviação. Um estagiário ouviu a palavra debug no meio da conversa e saiu-se com esta. Até hoje eu e esse diretor de informática usamos a expressão como piada local.

Bem, tentamos com cinco letras "A", a saída foi AAAAA. Com seis, AAAAAA. Com sete, AAAAAAA. Com oito, aconteceu algo interessante:

```
$ ./idiota
AAAAAAAA
AAAAAAAA
Falha de segmentação (core dumped)
$
```

Falha de segmentação! O programa falhou e gerou um relatório de erros com o conteúdo da memória, gravado no arquivo core. O que será que aconteceu? Lembremos agora da pilha. O espaço para nossa variável xuxu (que podemos chamar de buffer) era de 5 bytes – nós mesmos definimos isso (char xuxa[5]). Cada caracter ocupa 1 byte, portanto 5 caracteres enche o buffer. Mas o sistema não possui nenhum mecanismo de checagem, ou melhor, a linguagem C não possui. Portanto, se colocarmos mais de 6 caracteres no buffer, haverá estouro. Os dados a mais serão escritos por cima de alguma outra coisa (normalmente algo muito importante...).

Agora, o segundo detalhe. A pilha armazena de baixo para cima. Portanto, se você olhar a representação gráfica que fizemos dela, vai notar que o sexto caracter vai ser escrito sobre o Ponteiro da Pilha!

O ponteiro de pilha possui vários bytes (é de tamanho variável) e o de retorno, 4 bytes[3]. É por isso que, quando escrevemos 6 ou 7 caracteres, nada acontece – estamos sobrescrevendo o ponteiro da pilha, em alguma região que não nos afeta imediatamente. A partir do oitavo caractere temos um problema imediato: estamos sobrescrevendo alguma área importante do ponteiro de pilha.

Se avançarmos mais um pouco, lá pelo décimo-quarto ou décimo-quinto caracter certamente chegaremos ao ponteiro de retorno!!!!! Agora o primeiro byte do endereço de retorno não será mais o endereço original, será o valor hexa do caracter que digitamos! Se for A, por exemplo (ave Aleph1!) o valor hexadecimal será 41h. Vamos depurar nosso programa usando o gdb, um depurador GNU – para DOS, procure uma ferramenta adequada (como o debug) ou use uma das máquina Linux da nossa rede de testes. Primeiro, vamos rodar nosso programa e gerar um core dump com muitos "A"s:

```
$ ./idiota
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
Falha de segmentação (core dumped)
$
```

Agora, vamos rodar o gdb e ver o que ele nos diz:

```
$ gdb idiota core
GNU gdb 5.1
Copyright 2001 Free Software Foundation, Inc.
GDB is free software, covered by the GNU ...
(corta)
(no debugging symbols found)...
Core was generated by `./idiota'.
Program terminated with signal 11, Segmentation fault.
Reading symbols from /lib/libc.so.6...
(no debugging symbols found)...done.
Loaded symbols for /lib/libc.so.6
Reading symbols from /lib/ld-linux.so.2...done.
Loaded symbols for /lib/ld-linux.so.2
#0  0x41414141 in ?? ()
(gdb)
$
```

A linha importante, por enquanto, é `#0  0x41414141 in ?? ()`. Ela indica o endereço de retorno. Observe que, em vez do endereço correto, temos nosso caracter A (41h) repetido várias vezes. Bela maneira de fazer o programa abortar! Nosso programinha termina por ali mesmo, mas façamos um teste: entre as linhas funcao_idiota( ) e return 0; coloque a seguinte linha printf( ):

```
{
   funcao_idiota();
   printf("Nao atingiu Ponteiro de Retorno!\n\n");
   return 0;
}
```

Agora compile e rode o programa. Você verá que, até 11 caracteres, a mensagem final é mostrada antes do core dump. Com 12 caracteres ou mais, a mensagem é perdida: atingimos o ponteiro de retorno! Esse valor, claro, depende do programa, dos dados gravados no stack, do tipo de dados do nosso buffer, etc.

Ganhando acesso por estouro de pilha

Só que enfiar um bando de "A"s no buffer não é muito útil. E se pudéssemos executar algum código de máquina (sim, tem que ser código de máquina, afinal o programa está compilado...)? Poderíamos executar uma pequena rotina carregada no próprio buffer, que é o único lugar a que temos acesso. Desviar o processamento para o buffer não requer prática nem habilidade: já temos controle sobre o ponteiro de retorno, basta colocar nele o endereço do buffer em vez da montoeira de "A"s.

3. Explicação técnica avançada: os ponteiros têm quatro bytes de comprimento, ou 32 bits, o que equivale a um valor decimal entre 0 e 4.294.967.295, ou seja, 4 GB.

Note que este procedimento é bem parecido com a injeção de SQL que vimos no capítulo Vulnerabilidades II. Usamos um campo acessível externamente pelo usuário – no caso do SQL, um campo em um formulário; aqui, uma variável solicitando dados – e injetamos nele os comandos pertinentes.

Falamos em código de máquina, não é? Aqui temos o grande pulo-do-gato dos exploits que se aproveitam do buffer overflow de algum programa mal escrito. Nós queremos ganhar um shell no sistema, e isso depende do sistema operacional. O programa que oferece o shell num Solaris não é o mesmo de um MacOS que não é o mesmo no Windows... Por outro lado, para chamar esse shell temos que colocar um código de máquina no buffer, o que quer dizer que temos que fazer um exploit para cada dupla SO+Hardware existente. Um exploit para Linux em Alpha não é o mesmo que um para Linux em i386 (os PCs comuns). É o mesmo sistema operacional, mas não o mesmo processador. Por outro lado, um PC rodando Windows 2000 vai precisar de um exploit diferente do que um Solaris rodando no mesmíssimo PC. É o mesmo processador, mas não o mesmo SO. Mesmo considerando que estamos usando o mesmo programa bugado (por exemplo, o interpretador de Perl de cada um dos SOs) o exploit desse programa vai ser diferente em cada combinação SO+HW.

Para fins didáticos, vamos nos ater ao PC e ao Linux. Vamos "desassemblar" nosso programinha com o gdb:

```
$ gdb idiota
*** mensagens diversas ***
(gdb)disass main
Dump of assembler code for function main:
0x8048464 <main>:       push   %ebp
0x8048465 <main+1>:     mov    %esp,%ebp
0x8048467 <main+3>:     sub    $0x8,%esp
0x804846a <main+6>:     call   0x8048430 <funcao_idiota>
0x804846f <main+11>:    add    $0xfffffff4,%esp
0x8048472 <main+14>:    push   $0x8048520
0x8048477 <main+19>:    call   0x8048334 <printf>
0x804847c <main+24>:    add    $0x10,%esp
0x804847f <main+27>:    xor    %eax,%eax
0x8048481 <main+29>:    jmp    0x8048483 <main+31>
0x8048483 <main+31>:    leave
0x8048484 <main+32>:    ret
0x8048485 <main+33>:    lea    0x0(%esi,1),%esi
0x8048489 <main+37>:    lea    0x0(%edi,1),%edi
End of assembler dump.
(gdb)
```

Observe a linha

```
0x804846a <main+6>:     call   0x8048430 <funcao_idiota>
```


Desenferrujando um pouco nossos conhecimentos de assembler, lembramos que a função call chama uma outra função qualquer residente no endereço especificado. Como em linguagem de máquina não dá pra atribuir nomes às subrotinas, o sistema tem que saber exatamente onde na memória elas estão. No começo da listagem, observe que a função main inicia na posição de memória 0x8048464 e instrução call chama uma subrotina que se encontra em 0x8048430. Vamos dar agora uma olhada na função funcao_idiota( ):

```
(gdb) disass funcao_idiota
Dump of assembler code for function funcao_idiota:
0x8048430 <funcao_idiota>:      push   %ebp
0x8048431 <funcao_idiota+1>:    mov    %esp,%ebp
0x8048433 <funcao_idiota+3>:    sub    $0x18,%esp
0x8048436 <funcao_idiota+6>:    add    $0xfffffff4,%esp
0x8048439 <funcao_idiota+9>:    lea    0xfffffff8(%ebp),%eax
0x804843c <funcao_idiota+12>:   push   %eax
0x804843d <funcao_idiota+13>:   call   0x8048304 <gets>
0x8048442 <funcao_idiota+18>:   add    $0x10,%esp
0x8048445 <funcao_idiota+21>:   add    $0xfffffff8,%esp
0x8048448 <funcao_idiota+24>:   lea    0xfffffff8(%ebp),%eax
0x804844b <funcao_idiota+27>:   push   %eax
0x804844c <funcao_idiota+28>:   push   $0x8048500
0x8048451 <funcao_idiota+33>:   call   0x8048334 <printf>
0x8048456 <funcao_idiota+38>:   add    $0x10,%esp
0x8048459 <funcao_idiota+41>:   jmp    0x8048460 <funcao_idiota+48>
0x804845b <funcao_idiota+43>:   nop
0x804845c <funcao_idiota+44>:   lea    0x0(%esi,1),%esi
0x8048460 <funcao_idiota+48>:   leave
0x8048461 <funcao_idiota+49>:   ret
0x8048462 <funcao_idiota+50>:   mov    %esi,%esi
End of assembler dump.(gdb)
```

Olha só! O call da função main( ) chama exatamente a funcao_idiota(). Dentro da função idiota, a linha

```
0x8048461 <funcao_idiota+49>:   ret
```

mostra a instrução assembler de retorno (ret). Essa instrução vai usar o ponteiro de retorno para voltar ao main. Chegamos ao cerne da questão dos exploits por buffer overflow. Trocando em miúdos, você precisa:

1. Descobrir uma variável do tipo buffer que esteja vulnerável;
2. Verificar, no código fonte ou por tentativa e erro, os endereços onde as chamadas de função estão, bem como o endereço que marca o início do buffer da variável;
3. Fabricar um exploit que insira códigos de máquina no buffer, contendo

instruções para nos dar um shell, e depois "estufe" a pilha até atingir a posição do ponteiro de retorno, lá colocando o endereço do início do buffer.

Um exemplo, em Unix, seria uma rotina que, através da função execve( ) chamasse um shell. execve( ) é uma chamada de sistema que simplesmente permite executar um arquivo binário externo ao programa. Que beleza! De dentro de nosso exploit, executamos /bin/sh !!! O shell é executado no usuário do programa vulnerável. Se for, por exemplo, no Apache, ganharemos um shell do usuário nobody. O que hackers mais gostam, nessa hora, são programas vulneráveis rodando com SUID... Mas isso é outra história (vá lá e pesquise!).

No Windows, programas vulneráveis rodando com privilégio System são um perigo! Acesso total à máquina. Para explorar um buffer overflow, geralmente faz-se chamadas a funções de DLLs acessíveis pela aplicação vulnerável. Dica para estudo de execução arbitrária de comandos no Windows: WININET.DLL e o próprio MFC.DLL. Novamente, mexa-se...

Não vamos nos aprofundar mais, pois não está no escopo do livro entrar muito fundo nas entranhas de nenhum assembler. Para começar, um belo texto para iniciantes em buffer overflows (mixter.void.ru/exploit.html). Para saber mais, recomendo a leitura dos whitepapers da Fatal 3rror (struck.8m.com/f3), o excelente texto de Dark Spyrit sobre buffer overflows no Windows (community.core-sdi.com/~juliano/bufo.html) e o estudo que começou tudo isso: "Smashing the stack for fun and profit", do lendário Aleph1, lançado na edição 49 da ezine Phrak em 1996 e disponível online no endereço www.insecure.org/stf/smashstack.txt.

Outro white-paper digno de nota: como explorar serviços avançados com estouros de pilha, indo muito além de conseguir um shell. Taeho Oh nos mostra em postech.edu/~ohhara (ou, alternativamente, em ohhara.sarang.net/security/adv.txt) como furar firewalls baseados em filtros de pacotes, abrir sockets (e, portanto, backdoors) no próprio exploit e libertar-se da prisão do chroot[4].

Se você quer realmente conhecer a fundo todos os meandros dos estouros de pilha (nos quais 99% dos exploits existentes são baseados), recomendo, novamente, parar a leitura do livro e estudar os sites indicados. Lembre-se, o Google é seu amigo... Desnecessário dizer que é pré-requisito para isso saber alguns fundamentos de C e Assembler. Apesar de alguns dizerem que não, as linguagens Pascal (e portanto o Delphy/Kylix também) e Basic (Visual Basic, Turbo Basic, Quick Basic,...) e mesmo novas linguagens como C++, C# e Java também padecem do mesmo mal. As formas de explorar os estouros nessas linguagens são diferentes, mas ainda assim a vulnerabilidade existe.

4. Chroot é, simplificando a definição, uma maneira de "enjaular" uma aplicação que precise de direitos de superusuário para rodar. Normalmente, instala-se a aplicação em um diretório que possui uma cópia do sistema de arquivos do sistema operacional, mas não é o sistema de arquivos verdadeiro. Nesse ambiente falso, a aplicação roda com pseudo-direitos de root, que só valem dentro do ambiente. A aplicação fica feliz em ser enganada e roda perfeitamente. Se um hacker invadir essa máquina por meio de um buffer overflow da aplicação em chroot, vai conseguir, no máximo, o mesmo superusuário falso que a aplicação usa.

Ganhando acesso a outras contas

You know the day destroys the night / Night divides the day / Tried to run, tried to hide / Break on through to the other side[5]

Até agora, invadimos uma única máquina. Ok, você pode ter invadido diversas, mas mesmo que "0wne" dez ou quinze delas ainda não pesquisou como a rede ao redor funciona. Ou pior: você conseguiu um shell restrito e não consegue fazer muitas coisas! Encontramos dois exemplos clássicos nas páginas anteriores: o servidor Web Apache (que roda em usuário nobody) e aplicações rodando em chroot.

Tomemos o Apache: você aplicou um exploit nele e conseguiu um shell. Só que, nesse shell, o usuário que está logado é o nobody – um usuário especial criado justamente para não dar poderes especiais a potenciais invasores. Como o Apache não precisa de poderes especiais para rodar, apenas acesso a seus próprios arquivos, tudo corre às mil maravilhas. Um script kid entra por buffer overflow, tem acesso a um shell do nobody, pode, talvez, sobrescrever uma ou outra página HTML – e só! Não tem acesso à rede, não tem poderes de root, não possui sequer um diretório /home...

No Windows (NT e 2000), apesar de ser mais freqüente os administradores instalarem programas servidores nos grupos System ou Administrator, também é prática recomendada de segurança deixar tais programas com o mínimo possível de direitos sobre o sistema.

Nesse cenário, invadimos a máquina mas não temos, ainda, muito poder sobre ela. É hora, pois, de tentar conseguir, de alguma forma, acesso a outras contas e, se possível, ao superusuário.

A primeira forma é a que vimos nas páginas anteriores. Todo usuário tem acesso a diversos programas em sua máquina normal. Numa máquina Unix temos diversos scripts, programas de todos os tamanhos como fetchmail, MySQL, Informix, Oracle, sendmail, login, telnet, popd, inetd... Mais perto do usuário final, ainda, temos o servidor gráfico X Windows, os ambientes KDE, Gnome, CDE, WindowMaker (etc etc etc) e aplicativos associados. Temos ainda os próprios configuradores do sistema, como Linuxconf no Linux, Smit no AIX, Admintool no Solaris, SAM no HP-UX... Cada um deles com maior ou menor grau de direitos sobre o sistema. "Exploitar" qualquer um dos programas deste parágrafo pode levar ao root ou a, pelo menos, um usuário com mais direitos. Uma última dica: leia atentamente as páginas de manual dos comandos **su** *e* **sudo** *e do arquivo* **/etc/sudoers**. Você talvez ache interessante. (Hê hê hê...)

No Windows não é nada diferente. IIS, subsistema de segurança e login, Access, Word, Excel, Powerpoint (é triste, mas já encontramos servidores com o Office

5. "Quando o dia destrói a noite / a noite divide o dia / tentei correr, tentei me esconder / mas agora atravesso para o outro lado". Break on through [to the other side]. Do álbum The Doors, de 1967.

instalado...), MS SQL Server, CGIs diversos... Todos eles podem ser explorados para oferecer mais controle. Até que se chega em algum usuário ou programa que dá acesso à linha de comandos com os privilégios do grupo System (o "root" do Window NT). A dica, neste caso, fica por conta de exploits sucessivos (para ganhar cada vez mais poder) em direção a alguma DLL do próprio Kernel. Se você leu o capítulo sobre Plataformas Windows, aprendeu que várias partes do kernel rodam em User Mode. Algumas, mesmo assim, rodam com privilégios do grupo System. 2+2...

Embora seja efetivo e fácil, depender de exploits pode deixá-lo "na mão" em sistemas bem configurados e, principalmente, atualizados. Há, entretanto, formas alternativas de ganhar acesso a outras contas. A quebra de senhas é, seguramente, a mais usada.

Métodos para descobrir usuários e senhas

No capítulo 7 (Vulnerabilidades I) vimos diversas maneiras de quebrar as senhas do sistema. Vamos relembrar algumas:

1. **Logins fracos**: Duplas usuário/senha com palavras facilmente encontráveis no dicionário ou pior, que possam ser descobertas por engenharia social – são brinquedo na mão de hackers experientes. Listas de palavras e aplicativos que tentam combinações baseadas em dicionários existem às pencas por aí.
2. **Brute Force**: se os logins não são assim tão fracos, há ainda a possibilidade de direcionar um ataque "burro", que testa as combinações possíveis de letras, números e símbolos do teclado até encontrar uma ou mais duplas usuário/senha que entrem.
3. **Roubo e decifragem dos arquivos de senhas**: se o hacker conseguir obter uma cópia dos arquivos de senhas (SAM databases no Windows, /etc/passwd e /etc/shadow no Unix) está dentro! Basta usar as ferramentas conhecidas para tal (como o LophtCrack para Windows ou o John the Ripper para Unix) e pronto! Todas as senhas do sistema estão no papo, incluindo as do Administrador do Windows e do root no Unix.

Antes de tentar descobrir pares de logins e senhas, vamos ver um exemplo de ataque por dicionário e brute force. Observe uma coisa: já estamos dentro da máquina, portanto nosso script não vai mais enviar uma solicitação de login via rede. Podemos usar os procedimentos de login locais do próprio sistema operacional para tentar mudar de usuário. Novamente, um pouco de engenharia social, se possível, sempre ajuda.

Apesar de ser possível usar programas prontos para tal, como internamente cada caso é um caso o melhor caminho é fazer um script (em shell no Unix ou em WSE ou VBA no Windows) que teste seqüencialmente toda a sua lista de palavras em todas as combinações de logins e senha possíveis. Elaborar as listas com os nomes obtidos por engenharia social é uma boa prática.

Um script que fizesse isso deveria ter a seguinte estrutura:

1. Pega a próxima palavra do arquivo de dicionário;
2. Insere esse nome no programa de login do sistema;
3. Pega a primeira palavra do arquivo de dicionário;
4. Insere como senha no programa de login do sistema;
5. Pega a próxima palavra do arquivo de dicionário;
6. Insere como senha no programa de login do sistema;
7. Volta ao passo 5 até que todas as palavras tenhas sido usadas;
8. Volta ao passo 1 até que todas as palavras tenhas sido usadas.

É fácil implementar isso, por exemplo, com os programas login ou su do Unix em um shell script. Mas há um problema: VOCÊ ESTÁ SENDO VIGIADO!!! Todas as tentativas malsucedidas de login estarão sendo registradas nos logs do sistema. Como você ainda não tem privilégios de root, seria impossível apagar tais rastros. Esta, portanto, não é a melhor maneira de tentar entrar. Há uma, entretanto, fantasticamente simples e muito mais segura: o roubo do arquivo de senhas e posterior decifragem dos hashes.

É imperativo não ser descoberto. Portanto, a metodologia usada pela maioria dos crackers é obter o arquivo de senhas do sistema e tentar quebrá-las, offline, em casa. Pode-se usar diversos computadores ligados em clusters – é muito fácil fazer em casa, com Linux, supercomputadores com cinco ou seis 486s e mesmo 386s obtidos do lixo – e os programas quebradores de senhas já citados – L0phtCrack e John the Ripper. Uma única máquina rodando Windows NT com o L0phtCrack já é algo considerável: mesmo que demore um ou dois meses para conseguir alguma senha usável, tempo é o que o hacker mais tem. E, trabalhando em casa, o seu trabalho não será detectável.

Quebrando senhas no Windows

A sistemática é simples. Deve-se:

1. Roubar o arquivo de senhas e levar pra casa;
2. Passar o arquivo de senhas no programa quebrador;
3. Testar as senhas recuperadas no sistema original pra ver se os usuários não as alteraram.

Como exemplo prático, vamos usar um descendente do antigo L0phtCrack, o LC4, atualmente distribuído pela empresa de segurança @Stake (www.atstake.com). Criado por hackers como prova de conceito para demonstrar as fragilidades dos hashes do Windows NT, o software chegou até a versão 2.5 ainda com o código fonte aberto. No final da década de 90 os direitos sobre o software foram transferidos à atual proprietária, e o L0phtCrack 2.5 foi relançado como LC3. O LC4 é, portanto, uma evolução direta do L0phtCrack 2.5. O software é hoje vendido a um preço de US$350,00 por licença, mas é possível fazer download de uma versão de avaliação válida por 15 dias – com as rotinas de brute-force desabilitadas. No site, a @Stake oferece, gratuitamente e com código fonte, a versão 1.5 do L0phtCrack – ainda em linha de comando.

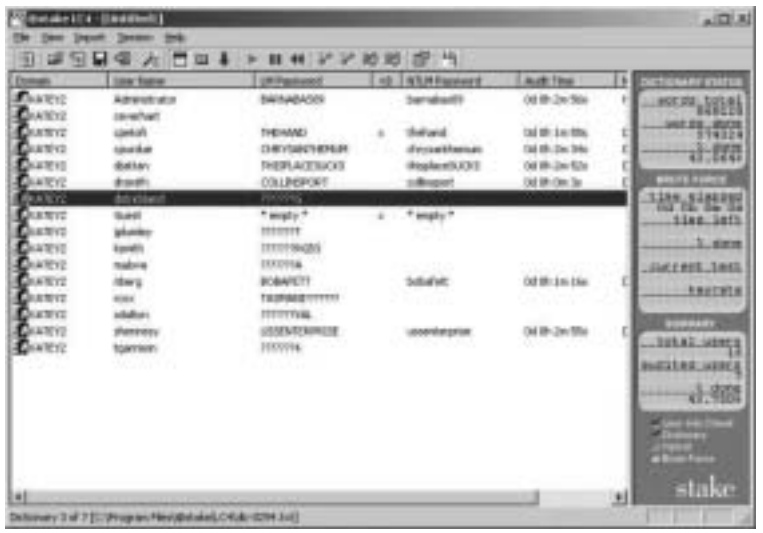
Para o passo 1, devemos de alguma forma "chupar" os hashes dos usuários contidos no banco de dados SAM, normalmente guardado no registro em **HKEY_LOCAL_MACHINE\SECURITY\SAM\Domains\Account\Users** (achou que não íamos dizer a chave?). Para tanto, podemos usar um utilitário do próprio LC4, chamado **pwdump**.

Outro programa que rouba as senhas do SAM é o pwdump3. Originalmente desenvolvido por Jeremy Allison and Todd Sabin, hoje é mantido por uma empresa, a PoliVec. Embutido em seu produto multiuso de segurança PoliVec Scanner, o pwdump3 (www.polivec.com/pwdump3.html) permite que as senhas sejam recuperadas inclusive remotamente, pela rede (incluindo a Internet).

Uma terceira opção é um utilitário que acompanha o pacote de servidores SMB para Unix, o Samba (www.samba.org). Junto com o produto está incluído um programinha chamado (olha que original...) **pwdump**, que extrai o SAM database e o transforma em um arquivo de senhas do Samba (**smbpasswd**) válido. Mais informações sobre ele podem ser obtidas em us1.samba.org/samba/ftp/pwdump/.

Nos três casos, gera-se arquivos de senhas que o LC4 pode interpretar. Nos três casos, também, precisa-se de privilégios de Administrador para rodar o programa. Esse obstáculo, apesar de complicado, ainda assim é contornável. Há diversas maneiras de fazê-lo: rodando o programa na máquina alvo através de um exploit, rodando o programa em uma máquina de senhas fracas com relações de confiança com a máquina alvo, fazendo invasão física no local onde a máquina está...

Uma vez conseguido o arquivo de senhas, passemos ao passo 2. Desconecte-se completamente da rede a ser invadida e rode, em casa, o LC4 contra seu arquivo. Há uma farta documentação incluída com o próprio produto, portanto não vamos detalhá-lo aqui. A figura a seguir mostra um processo de cracking em andamento, com algumas senhas já descobertas:



Observe: como vimos nos capítulos sobre Vulnerabilidades, as contas no WinNT são guardadas em 2 hashes: um para o antigo Lan Manager e outro com a senha mais forte do NT. Além do LM Hash ser mais fácil de quebrar, normalmente os usuários usam a mesma senha localmente (NTLM). O próprio LC4 sabe disso e, uma vez conseguida a senha LM, interrompe o brute force e tenta a mesma senha no NT. Como você pode ver na figura, várias senhas já estão descobertas e, no exemplo, TODAS são iguais nos dois campos. O usuário shennesy parece gostar do seriado Star Treck e o usuário djattan parece não estar muito contente com seu local de trabalho - com um pouco de engenharia social, o hacker nem precisaria do LC4. O usuário Guest não possui senha (glup!).

A figura anterior foi retirada de um estudo de Dan Smith, na Universidade da Carolina do Norte, EUA, e pode ser encontrado em *www.unc.edu/~smithdr/inls187/sr.html* . O estudo, além de analisar o software, traz dados sobre tempo para quebrar senhas de todos os tipos, fracas ou teoricamente seguras.

Por último (passo 3), teste todas as senhas que conseguiu no sistema invadido. É possível que algumas estejam trocadas, mas você já sabe, pelo menos, o nome dos usuários válidos e o padrão que eles usam para elaborar suas senhas. Por exemplo, a conta Administrator mostrada possuía uma senha BARNABAS09. As chances estão entre 80% e 90% de que, nos próximos meses, ela seja BARNABAS10, BARNABAS11... Acho que o leitor já "pescou" o mecanismo!

Quebrando senhas no Unix

Você acha que a sistemática mudou? Nada disso:

1. Roubar o arquivo de senhas e levar pra casa;
2. Passar o arquivo de senhas no programa quebrador;
3. Testar as senhas recuperadas no sistema original pra ver se o usuário não a alterou.

Há diversos programas para Unix que descobrem senhas em arquivos criptografados. O mais conhecido certamente é o John the Ripper. Usá-lo é tão fácil que dá raiva. Uma vez instalado e configurado, basta "alimentá-lo" com o arquivo de senhas e deixar que ele faça o serviço. A sintaxe é simples:

```
$ john arquivo_senhas
```

No passo 1 temos os mesmo problemas que tivemos com o Windows. Como vimos no capítulo sobre Plataformas Unix, os sistemas modernos utilizam o sombreamento de senhas, que consiste em separar o processo de login em dois arquivos: o /etc/passwd (que contém informações gerais sobre o usuário, como nome, login, GID, UID, diretório /home e shell inicial) e o /etc/shadow (que contém as senhas criptografadas). Como o /etc/passwd tem que ser legível por todos, os hashes ficariam desprotegidos se ficassem aí. O arquivo /etc/shadow, ao contrário, é legível e editável apenas e tão somente pelo superusuário, o root. Se o sistema invadido possuir sombreamento de senhas, você só conseguirá roubar o arquivo /etc/shadow se estiver em um shell root. Se o sistema invadido não possuir sombreamento, o "administrador" já deixou metade do serviço pronto para você...[6]

6. Mesmo que o sistema possua password shadowing, você ainda terá os nomes válidos de contas.

Encontrar e aplicar um exploit que lhe dê acesso a root é a forma mais direta para conseguir o arquivo de shadow. Outras formas incluem:

1. Fazer core dumps de programas SUID root que acessem as senhas (como o FTP, o SSH ou o Telnet);
2. Verificar no arquivo passwd qual dos usuários têm "pinta" de ad mistrador – é possível que ele possua programas SUID em seu /home.

Uma vez descoberta a senha, chegamos ao passo 2. Basta rodar o John the Riper de acordo com o configurado e esperar. No final, uma lista de senhas válidas (possivelmente a do root também) será encontrada. O passo 3 é trivial: volte à cena do crime e use as senhas. Nada como logar-se como um usuário autorizado para despistar as auditorias de invasão...

Ganhando acesso e destruindo a rede

Uma vez dentro da rede, há diversas ações possíveis de ser tomadas. Estando em posse de uma máquina, pode ser mais fácil invadir outras. Mas nem só de "0wnar" vive um cracker. É possível, por exemplo, usar a rede como trampolim para outras redes maiores. Ou capturar senhas dos usuários em serviçoes externos – como senhas de bancos ou números do cartão de crédito. Ou ainda desviar o tráfego para que os usuários sejam encaminhados para armadilhas ou sites falsos.

Apresentaremos nesta seção apenas descrições resumidas desses tipos de ataques, com indicações para sites com mais informações.

War Driving e War Chalking

Uma das mais novas formas de invasão corporativa é o War Driving. Hackers saem pelas ruas da cidade com gambiarras feitas de latas de batata frita, arruelas e alguns cabos, e capturam conexões de rede wireless que estejam "vazando". Como as corporações não costumam criptografar suas conexões internas, uma conexão wireless desse tipo fornece um acesso ilimitado, similar ao conseguido se o invasor entrar andando pela porta da frente e plugar um cabo do seu laptop em uma tomada de rede da empresa.

O assunto é relativamente novo e merece um estudo muito mais profundo do que o que seria permitido neste livro. Aliás, seria necessário um livro inteiro sobre isso. Uma excelente fonte de informações e recursos (tutoriais, equipamentos, esquemas de antena – inclusive feitas em casa) é o site www.wardriving.com. Um esquema simples de antena, construído com o já lendário tubo de batatas fritas Pringles, pode ser encontrado em www.oreillynet.com/cs/weblog/view/wlg/448.

Hackers europeus foram mais longe e criaram o War Chalking – um alfabeto especial para marcar, com giz e na calçada, os pontos de acesso (posição e orientação da antena) para uma melhor conexão à rede alheia. Saiba mais sobre as técnicas usadas no site oficial: www.warchalking.org (ou, alternativamente, www.warchalking.us). Esse site traz informações técnicas, tutoriais, clubes e muitos links para outros recursos. O site www.blackbeltjones.com também tem diversos recursos. Um artigo sobre o assunto pode ser encontrado na Meca do wireless, o site 802.11 Planet (www.80211-planet.com/columns/article.php/1402401).

Indo além do SQL Injection...

Em nosso segundo estudo sobre vulnerabilidades, vimos maneiras simples de enganar o script da página (seja em ASP, PHP, ColdFusion ou algum CGI) e injetar nela modificações na query SQL associada. Mas há outras maneiras de brincar com sistemas baseados em Web.

A primeira e mais imediata é observar e manipular os dados da própria URL. Muitos sistemas on-line carregam informações valiosas em campos presentes depois do endereço HTTP (tarefa para casa: pesquisar sobre os métodos HTML GET e POST). Pentear uma URL grande tentando entender como o sistema funciona é o primeiro passo para entrar em tal sistema. Bancos, sites de compras e mesmo sites governamentais utilizam a URL como transporte de dados do cliente sem se importar com o fato de serem informações visíveis por qualquer um na rede.

Um exemplo notório disso foram as vulnerabilidades encontradas há alguns anos no Hotmail (www.hotmail.com), em que os identificadores de sessão, nomes de usuário e hashes das senhas estavam presentes na própria URL. Hoje o Hotmail já não padece mais desse mal, mas por muitos anos foi uma brecha extremamente fácil de explorar.

Preste atenção ainda em sites que usam chamadas ODBC e não SQL. Sistemas 100% Microsoft tendem a privilegiar o inseguro protocolo ODBC para acesso a bancos de dados Access e SQL-Server.

Alguns links para pesquisa sobre SQL Injection:

- www.securiteam.com/securityreviews/5DP0N1P76E.html
- online.securityfocus.com/infocus/1644
- www.sqlsecurity.com/DesktopDefault.aspx?tabindex=2&tabid=3
- www.nextgenss.com/papers/advanced_sql_injection.pdf
- www.spidynamics.com/papers/SQLInjectionWhitePaper.pdf

Outro método é o ***cookie poisoning***, uma maneira de torcer a autenticação via W eb. Quase a totalidade dos sites hoje em dia utilizam-se de cookies para controlar acesso e sessão. Alterar UserID e SessionID em cookies pode ser um atalho para entrar em contas de outros usuários.

É mais difícil encontrar recursos sobre cookie poisoning na Internet. Sugerimos os seguintes recursos:

- White paper: Hacking web applications using cookie poisoning (www.allasso.pt/base/docs/11042206054.pdf).
- Busca pela palavra chave cookie no Security Focus (www.securityfocus.com).

Dois softwares que trabalham como Proxy baseados no desktop podem ser usados para facilitar a manipulação de Cookies (e de cabeçalhos HTTP também!): são eles o Achilles e o BrowseGate. A empresa que produz o Achilles (DigiZen Security Group – www.digizen-security.com) parece ter retirado o site do ar, mas descrições sobre o produto podem ser encontradas no PacketStorm (packetstormsecurity.nl/filedesc/achilles-0-27.zip.html) e no SecuriTeam.com (www.securiteam.com/tools/6L00R200KA.html).

O BrowseGate, desenvolvido pela NetCPlus (www.netcplus.com/browsegate.html) é outra opção de Proxy Server que pode ser usada de maneira maléfica para torcer cookies e autenticação em sites. Há uma análise dele em www.webattack.com/get/browsegate.shtml.

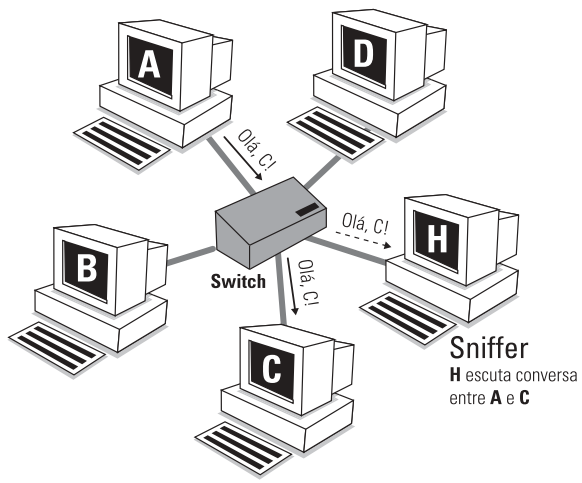
Para todos os problemas listados, possíveis soluções podem ser encontradas no site oficial sobre segurança em sistemas web: www.owasp.org.

Farejando a rede (Sniffing)

Outra forma de ganhar acesso a uma rede, depois de "0wnar" uma das máquinas que a integram, é passar a "escutar" o que está trafegando nessa rede. Os programas que fazem esse tipo de trabalho sujo são chamados de sniffers ou farejadores.

Um sniffer trabalha na camada 2 de nosso modelo de referência OSI. Isso quer dizer que é impossível fazer um sniffing diretamente via Internet em uma rede distante. É necessário que o invasor instale e rode o sniffer em uma máquina pertencente à rede local que se quer farejar.

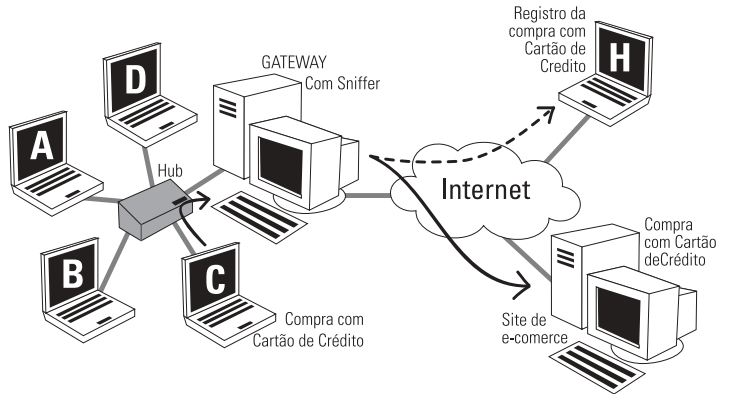
O objetivo mais imediato de um invasor quando instala um sniffer é descobrir senhas de outros usuários da mesma rede. Deixando a ferramenta "de campana" por alguns dias, é possível conseguir senhas de dezenas de usuários e centenas de serviços (sites, e-mail, servidores, sistemas...). Como há serviços autorizados para alguns usuários em especial e negados aos outros, é interessante, à medida que se vá conseguindo senhas, que instale sniffers em diversas máquinas e, com isso, consiga um universo maior delas.



É fácil perceber que, nesse contexto, o invasor vai "0wnando" aos poucos um grande número de máquinas dentro da rede remota. Como é uma ferramenta da camada 2 e, portanto, local, o sniffer deve ser instalado e deixado em atividade sem que o invasor interve-

nha. A ferramenta escutará a rede e gravará tudo o que for de interesse em um arquivo. Depois de algum tempo (alguns dias ou semanas) o hacker voltará ao local do crime apenas para recuperar o arquivo com o tesouro, o qual analisará em casa, desconectado.

Há uma forma mais perniciosa de usar o sniffer: colocá-lo em um gateway entre redes. Como vimos nos capítulos pertinentes, um gateway é um equipamento que une duas ou mais redes diferentes de forma a passar pacotes entre elas quando aplicável. Um sniffer colocado num gateway pode escutar, então, o tráfego de todas elas.



Na prática, como a maioria esmagadora dos gateways ligam sua rede interna à Internet, o que o invasor tem à disposição é tanto seu conjunto potencial de senhas e informações confidenciais quanto os e-mails, senhas, informações e cartões de crédito que entram e saem de sua empresa. Imagine, num cenário ainda mais tenebroso, que um hacker plantou um sniffer em um gateway que liga sua empresa ao seu site de comércio eletrônico, ou à operadora de cartões de crédito, ou a seu parceiro, ou ao banco. Fatal!

A primeira providência quando se vai farejar uma rede é colocar a interface de rede de alguma máquina em modo promíscuo. Como vimos nos capítulos Redes I e II, quando um pacote IP chega em uma rede, a interface que detém o pacote pergunta: "qual é o MAC Address que contém o endereço IP desse pacote"? A máquina destino responde com seu MAC Address e o pacote é enviado a ela. Esse é o conhecido protocolo ARP.

"Enviado a ela", como escrito no parágrafo anterior, é ligeiramente mentiroso (ou, como dizia um pastor luterano que conheci, é um "exagero da verdade"). O pacote é, na verdade, jogado no barramento e todas as interfaces podem ler. O que ocorre é que as interfaces fazem "ouvidos moucos" ao pacote, caso este não seja direcionado a elas. Apenas a máquina a que realmente se destina "presta atenção" ao que está trafegando na rede. As outras simplesmente ignoram-no.

É aqui que entra o "modo promíscuo". Uma interface configurada dessa forma "ouve" TODOS os pacotes que trafegam na rede, e não apenas os que são destinados

a ela. Se isso é um facilitador para a implementação de ferramentas de monitoração de rede – coisa que todo administrador competente deveria usar – também possibilita que alguém com más intenções facilmente escute o que não devia.

Para colocar uma interface de rede em modo promíscuo, deve-se ter acesso privilegiado ao sistema operacional – o que equivale a dizer **root** em um sistema Unix, **Administrator** em um sistema WinNT ou **Admin** em um Novell Netware. Por isso mesmo, tomar completamente pelo menos uma máquina na rede (como vimos anteriormente) é imperativo para que possamos fazer uma "colheita" posterior. Há diversas formas de colocar uma interface de rede em modo promíscuo. Há programas especiais para isso em qualquer plataforma e mesmo alguns truques no sistema operacional (seja ele Windows, Novell, HP-UX...) são possíveis sem o auxílio de qualquer programa externo.

Todo sistema operacional possui uma ferramenta qualquer que mostra o estado e altera as configurações da placa de rede.Tomando como exemplo um sistema GNU/Linux, o comando que faz esse trabalho para nós é o **ifconfig**. Emitido (como root) sem argumentos, o comando mostra o estado de todas as interfaces de rede. Emitido seguido do nome de uma interface, dá o estado atual dessa interface. Por exemplo, para saber a quantas anda a primeira interface de rede (eth0), o comando seria:

```
# ifconfig eth0
```

O resultado seria:

```
Encapsulamento do Link: Ethernet  Endereço de HW 00:08:74:B5:64:95
inet end.: 192.168.1.11  Bcast:192.168.1.255  Masc:255.255.255.0
UP BROADCAST RUNNING MULTICAST  MTU:1500  Métrica:1
RX packets:13567 errors:0 dropped:0 overruns:1 frame:0
TX packets:8300 errors:0 dropped:0 overruns:0 carrier:0
colisões:0
RX bytes:3163776 (3.0 Mb)  TX bytes:994079 (970.7 Kb)
```

Observe as informações mostradas.Você sabe, por meio desse comando, que o encapsulamento do link (ou seja, o protocolo de camada 2) é Ethernet, que o MAC é 00-08-74-B5-64-95, que o endereço de rede é 192.168.1.11, que o tamanho máximo do pacote Ethernet (MTU) é de 1.500 bytes, etc... Há também alguns flags que indicam se a interface está pronta ou "em pé" (UP), se está rodando (RUNNING) e se responde a broadcast ou multicast.Agora vejamos o que acontece com o comando abaixo:

```
# ifconfig eth0 promisc
```

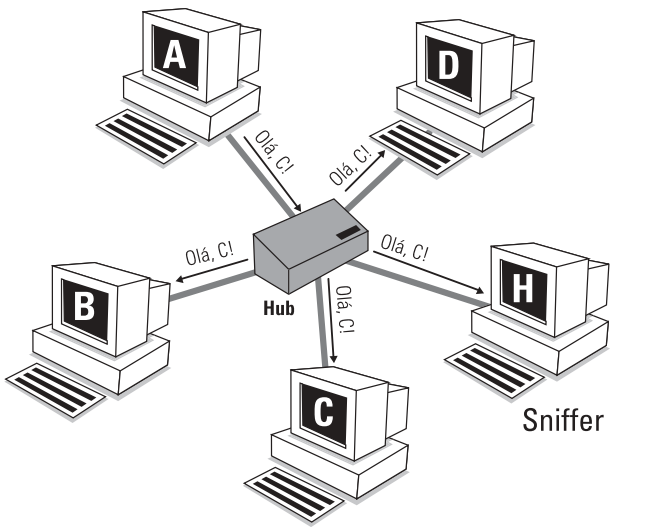
Aparentemente, nada acontece. O shell nos devolve o prompt e nenhuma mensagem de erro ou de tarefa conlcuída é mostrada. Mas, se emitirmos novamente o comando **ifconfig eth0**, o resultado seria um pouco diferente:


```
Encapsulamento do Link: Ethernet  Endereço de HW 00:08:74:B5:64:95
inet end.: 192.168.1.11  Bcast:192.168.1.255  Masc:255.255.255.0
UP BROADCAST RUNNING PROMISC MULTICAST  MTU:1500  Métrica:1
RX packets:13567 errors:0 dropped:0 overruns:1 frame:0
TX packets:8300 errors:0 dropped:0 overruns:0 carrier:0 colisões:0
RX bytes:3163776 (3.0 Mb)  TX bytes:994079 (970.7 Kb)
```

Bingo! Observe que a placa agora está em modo promíscuo (flag PROMISC). Absolutamente TUDO o que está trafegando na rede é interpretado pela pilha TCP/IP do kernel e pode, portanto, ser monitorado. Isso inclui pacotes não direcionados a essa máquina.

Sniffing passivo

As redes locais normalmente se utilizam de equipamentos chamados hubs (capítulos Redes I e II) para facilitar e flexibilizar a conexão de novas máquinas a uma rede existente. O hub age ainda como um elemento regenerador do sinal elétrico presente no barramento de rede. Mas o hub é um elemento **passivo** no tocante a controle do tráfego na rede local.

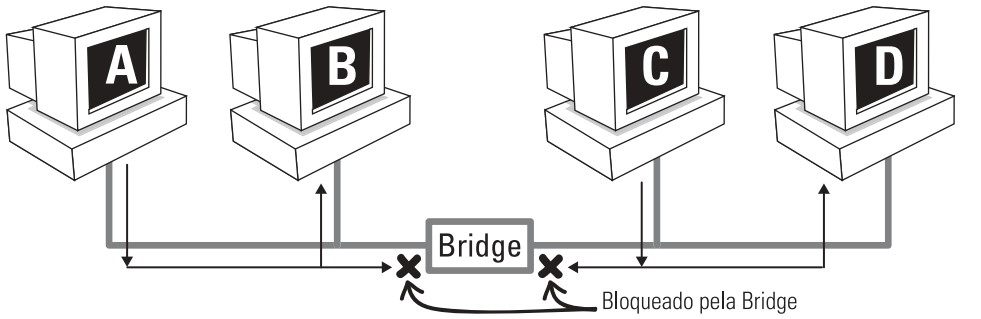


Hubs e repetidores são equipamentos que trabalham na camada 1 do modelo OSI, portanto não têm controle algum sobre o quadro Ethernet (ou qualquer outro protocolo de camada 2 como Token Ring, PPP, Frame Relay ou X.25). Isso significa que uma mensagem enviada de uma máquina para a outra será ouvida por todas na rede.

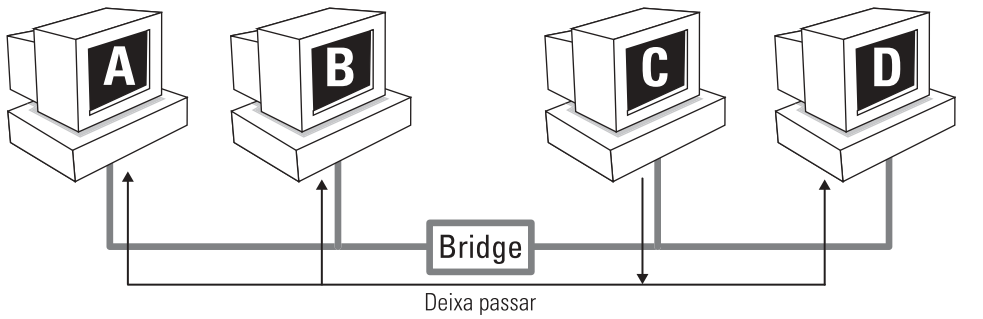
É fácil perceber, no desenho acima, que se a máquina A enviar uma mensagem para a C, as estações B, D e H ainda estarão "ouvindo". Portanto, apesar da aparência de estrela, uma rede que se utilize de um hub para interligar as máquinas é, na realidade, um barramento.

Concluímos que é muito fácil para um sniffer registrar e decodificar tudo o que trafega nessa rede. Como trabalha com interfaces em modo promíscuo, todos os pacotes da rede podem ser interpretados. Mas e se a rede estiver segmentada com bridges ou switches?

No capítulo 2 (Redes I) vimos en passant a descrição desses equipamentos. Uma bridge divide a rede em dois segmentos e bloqueia tráfego não destinado cada um deles.

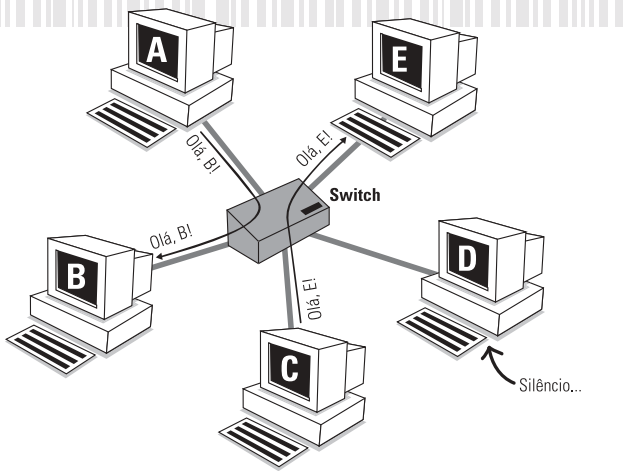


Observe: as máquinas A e B estão no segmento 1, enquanto C e D no segmento 2. Isolando os dois, uma bridge. O tráfego de A para B e de B para A fica restrito ao segmento 1, e o tráfego de C para D e de D para C fica restrito ao segmento 2. A bridge bloqueia o tráfego local, não deixando que mensagens não endereçadas a um barramento em especial cheguem nele.



Entretanto, se a máquina C quer enviar uma mensagem para a máquina A, a bridge deixa passar o pacote.

A determinação de quais pacotes devem passar e quais devem ser bloqueados é dinâmica, baseada nos endereços MAC das interfaces de rede. No momento que uma bridge é ligada, ela não tem nada na memória. À medida que as estações vão enviando pacotes para a rede, a bridge guarda os MAC Adresses numa tabela, relacionando-os com o segmento de onde o pacote se originou. Note que tudo se passa na camada 2 do modelo OSI: a bridge só tem conhecimento das máquinas ligadas diretamente em sua rede local.



Imagine, agora, um hub que possuísse uma bridge em cada porta. Cada máquina da rede receberia apenas tráfego destinado a si. Esse "monstrengo existe e se chama comutador ou switch. Observe: assim como em nosso primeiro exemplo, a estação A quer enviar uma mensagem para a estação B. Devido ao switch, nenhuma das outras máquinas irá escutar o que A tem a dizer. Além disso, C poderia falar com E simultaneamente, pois a rede está, para elas, ociosa.

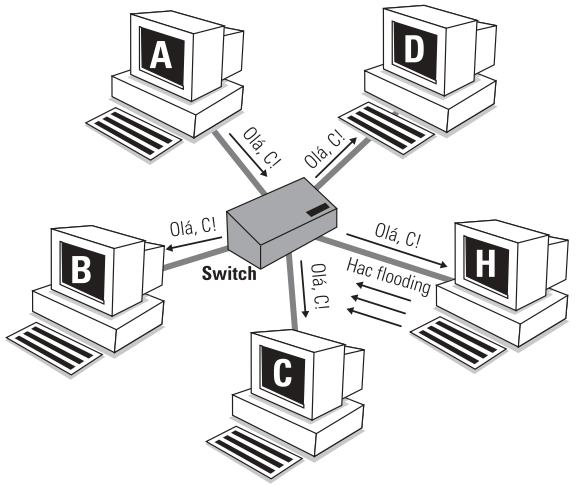
Uma rede com um switch no lugar de um hub, além do controle e diminuição do tráfego, também ofereceria uma segurança adicional à rede, uma vez que um sniffer instalado, por exemplo, em D não poderia escutar nada das conversas entre A e B ou C e E. O uso de bridges e switches, então, minimizaria o problema dos sniffers, certo? Infelizmente, mais uma vez, a resposta é não...

Sniffing ativo

Um switch ou bridge possui uma tabela que relaciona os MAC addresses que "ouviu" na rede com as portas ou segmentos nos quais foram "ouvidos". Como é preenchida dinamicamente, essa tabela será atualizada toda vez que uma nova máquina for conectada à rede.

Como dizia um parente meu (ilustre, mas iletrado), "tudo o que é demais é em demazia" (sic). A memória do switch possui um tamanho limitado, portanto um número muito grande de interfaces de rede ligadas em cada uma das portas poderia, num caso extremo, preenchê-la completamente. Pensando nisso, os fabricantes desses equipamentos as dimensionam para que esse limite nunca seja atingido.

O problema é que um quadro Ethernet nada mais é do que uma seqüência de uns e zeros que pode, por isso mesmo, ser manipulada. Um programa cuidadosamente escrito poderia gerar, ad infinitum, frames Ethernet com MAC adresses aleatórios e, em alguns minutos, preencher completamente a memória do switch. Como o show não pode parar, esses equipamentos podem começar a repassar mensagens indiscriminadamente para todas as portas. Essa técnica é chamada de MAC Flooding.

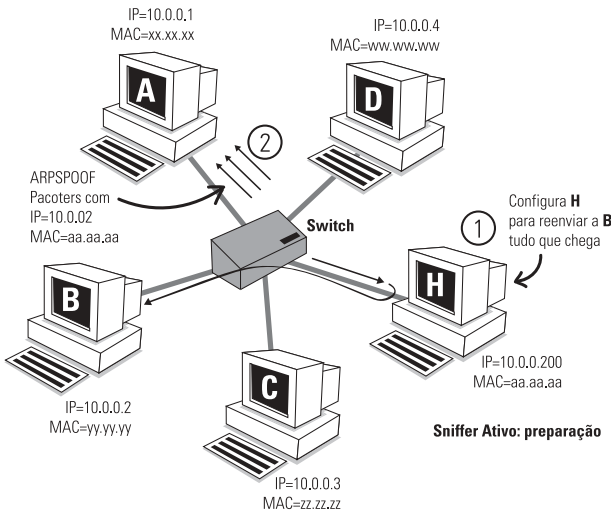


Uma vez nessa situação, qualquer sniffer poderá farejar a rede. Há ferramentas que fazem isso (uma delas, escrita em Perl, pode ser encontrada em www.safenetworks.com/Others/3com4.html), mas os sniffers mais modernos (como o ettercap e o dsniff) já fazem todo o trabalho.

Felizmente (ou infelizmente, dependendo dos intentos do estimado leitor), alguns switches são imunes ao MAC Flooding. Há várias maneiras de implementar switches assim. Podem ser usados algoritmos de proteção que impedem que a memória seja completamente preenchida. Ou então um sistema de detecção de flood, baseado em padrões de dados e um knlowledge base dos sniffers conhecidos. Uma terceira maneira seria adotar uma política de persistência, mantendo MACs conhecidos há mais tempo em detrimento de novas interfaces, caso a memória lote.

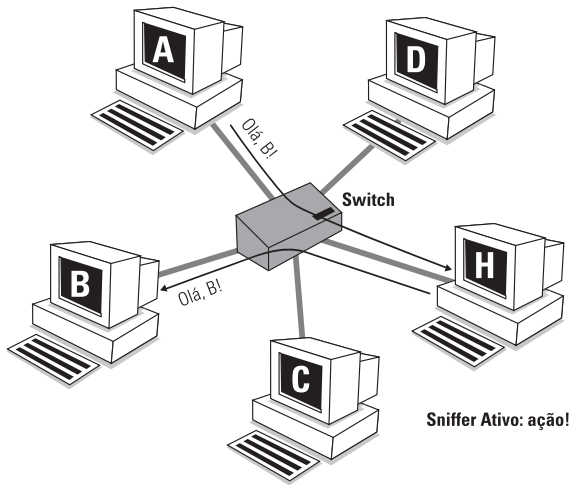
Para contornar esse inconveniente, os hackers desenvolveram uma técnica chamada **ARP Spoofing**. É uma técnica um pouco mais complicada, mas muito inteligente. Em vez de trabalhar apenas na camada 2 (Ethernet) o invasor vai confundir o computador cujo tráfego se deseja "esnifar" manipulando sua tabela ARP.

O ARP, como discutido anteriormente, é um mecanismo de tradução IP-para-MAC. A máquina que quer enviar o pacote pergunta, na rede: "Quem tem o IP tal"? Todas as máquinas daquele segmento ouvem a pergunta, mas apenas a interface que possui aquele IP responde: "Sou eu, meu MAC é XXXXXX". A partir disso, a interface de origem monta um quadro Ethernet e o envia ao destino. O ARP Spoofing é uma maneira de enganar a máquina da vítima, fazendo-a acreditar que o endereço MAC da máquina onde o sniffer está corresponde ao endereço IP da máquina-destino original. Complicado? Sim, é. Vamos tentar exemplificar:



No diagrama mostrado, a estação A quer falar com a estação B. A partir da estação H, um hacker quer farejar a comunicação entre A e B. Para isso, é necessário uma preparação prévia. A primeira coisa que o hacker deve fazer é configurar seu sistema operacional de H para repassar todo e qualquer tráfego que chegue para si, vindo de A, para a verdadeira máquina destino, que é B. A configuração do **IP forwarding** normalmente é feita pelo prórpio sniffer, mas é possível que em alguns casos seja necessário fazê-lo manualmente. Tanto Windows quanto Netware e também qualquer Unix permitem esse tipo de redirecionamento. Lembra-se daquele mapa da rede, feito com ping+traceroute (ou com o Cheops)? Ele será muito útil agora.

O segundo passo é enganar a máquina A, fazendo-a acreditar que o IP de B possui o MAC de H. Isso é conseguido fazendo H enviar um número monstruoso de respostas ARP para A, informando que o IP de B possui o MAC de H – respostas essas que sequer foram solicitadas. Depois de um certo tempo, A "pensa" que, para mandar mensagens para o IP de B, tem que construir quadros Ethernet direcionados ao MAC de H.



Agora é só ativar seu sniffer preferido e esperar. O tráfego vindo de A em direção a B vai passar por H antes. Nem A nem B vão desconfiar disso, pois, para eles, a comunicação é apenas entre os dois. Esse tipo de configuração de ataque é normalmente chamado de "man in the middle".

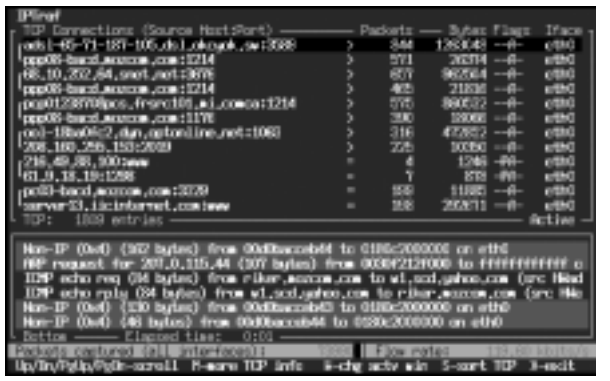
Escolhendo seu sabujo

Há um número gigantesco de ferramentas de sniffing que podem ser usadas em diversas plataformas. Eis alguns:

- **tcpdump** (www.tcpdump.org), a ferramenta nativa de monitoramento de rede de qualquer Unix. Já vimos o funcionamento do tcpdump no capítulo Redes II. Colocando a placa de rede em modo promíscuo (por exemplo, com ifconfig no Linux) e rodando o tcpdump, tem-se o mais básico possível (mas não por isso menos eficiente) sniffer de rede para Unix. A vantagem de se usar o tcpdump é que todos os Unices já o possuem – o invasor não precisa instalar nada, só rodar o programa e direcionar a saída para um arquivo. O comando seria:

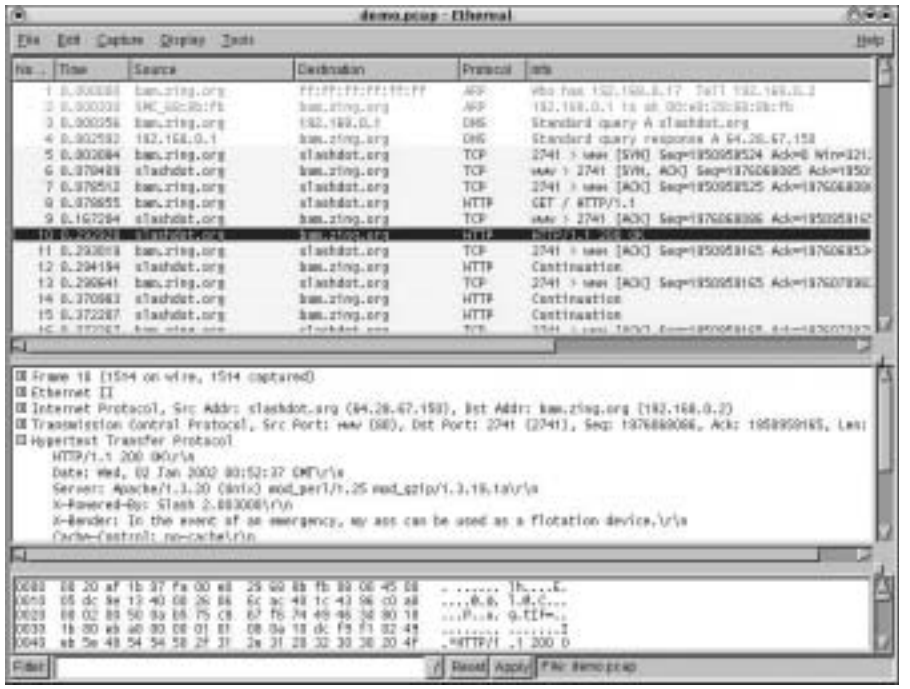
tcpdump [opções] > arquivo.de.saída

Uma versão do tcpdump para Windows pode ser encontrada em **windump.polito.it**.



- **IPTraf** (iptraf.seul.org), o avô de todos os sniffers Unix, ainda pode ser efetivamente usado em uma sessão de sniffing. Apesar de sua idade, possui recursos poderosos. Vale a pena dar uma conferida. Uma de suas vantagens é que, como o tcpdump, é comum encontrá-lo já instalado – poupando trabalho, recursos e não levantando suspeitas.

- **Ethereal** (www.ethereal. com), um sniffer poderoso, que suporta vários protocolos, marca-os com cores diferentes e interpreta seus significados. Possui uma interface gráfica muitíssimo amigável. É mais usada por administradores, mas é útil também para crackers malintencionados. Disponível para Windows e Unix. Os autores recomendam.



O Ethereal pode, inclusive, decodificar e extrair informações a partir de um determinado protocolo. No exemplo acima, uma página HTML foi decodificada de dentro de uma série de pacotes TCP:

- **Sniff'em** (www.sniff-em.com), um sniffer comercial para Windows. Faz sozinho tudo o que o Ethereal, o Ettercap, o Snort e o Dsniff fazem juntos, mas é pago. Na opinião dos autores, não valçe a pena pagar para ter algo que pode ser obtido gratuitamente com ferramentas menores, mas usuários de Windows gostam de ter tudo integrado num único programa. Um dos diferencias do Sniff'em é sua capacidade de monitorar interfaces não-usuais como saídas seriais, USB e modems RAS e RDSI/ISDN.

- **Snort** (www.snort.org) é um detector de intrusos (IDS) que serve também como sniffer. É muito conhecido por analisar de forma perfeita os logs do Squid. Tem a seu favor, além de sua coleção de truques, uma vasta lista de plataformas na qual roda, incluindo aí Linux, Solaris, *BSD, HP-UX, IRIX, MacOS X, AIX e Windows.

- **Sniffit** (reptile.rug.ac.be/~coder/sniffit/sniffit.html) trabalha exclusivamente em Unix, mas é venerado pela comunidade por suas capacidades de sniffing quase esotéricas. Possui dois modos de operação. O tradicional (sniff mode) faz o que qualquer sniffer faz: grava todo o tráfego da rede. Um segundo modo, chamado de interativo, permite que o hacker veja o que está trafegando na rede em tempo real. É possível escolher apenas um protocolo para monitorar, e existem filtros que escondem toda a complexidade inerente dos protocolos, mostrando apenas os dados úteis. Com o sniffit, é possível inclusive ver o que a vítima está digitando em um programa de mensagens instantâneas como o MSN Messenger ou o ICQ – tudo isso em tempo real!

▸ **Hafiye** (www.enderunix.org/hafiye) é um sniffer baseado em knowledge-base. Possui um banco de dados com padrões dos mais diversos protocolos de comunicação e criptografia e é capaz de separar os dados úteis de dentro de um pacote.

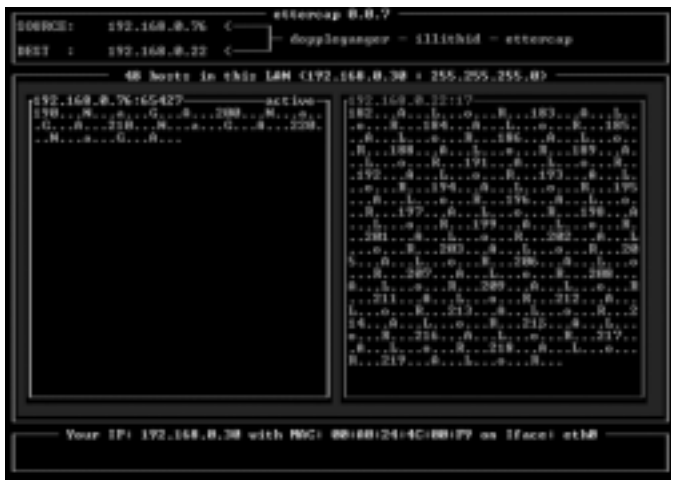
▸ **Kismet** (www.kismetwireless.net) é um farejador de redes wireless (802.11) nos moldes do IPTraf e tcpdump. Além de ser capaz de decodificar os protocolos pertinentes, consegue dividir as redes por área e por célula de rádio. Desenvolvido especialmente para Linux, suporta nativamente o PDA Zaurus, da Sharp, com placa de rede wireless. Uma boa pedida para War Driving.

▸ **USB Snoop** (sourceforge.net/projects/usbsnoop) monitora e grava todo o tráfego entre o driver do dispositivo USB e o próprio dispositivo. Útil para engenharia reversa de drivers (para, por exemplo, produzir um driver Open Source), mas também pode ser usado para monitorar o tráfego de modems USB. Uma rede criptografada e superprotegida pode ser "descadeirada" se um laptop for ligado a um computador de mesa ligado a ela pelo USB.

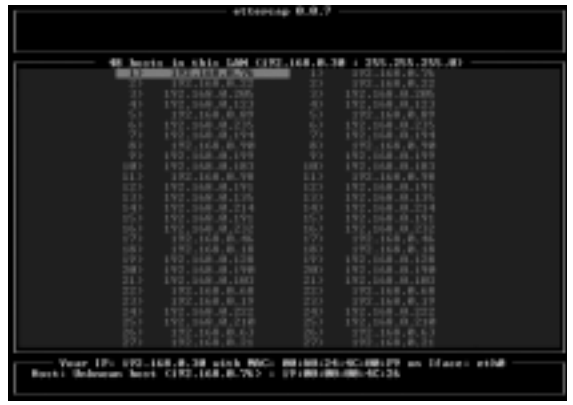
▸ **APS** – Advanced Packet Sniffer (www.s-wrtec.de/clinux) é outro exemplo de programa minúsculo que pode ser usado para farejar pacotes. Como é extremamente dependente do kernel, funciona apenas em Linux, mas sua interface simples e intuitiva em modo texto permite sua operação remota sem sobrecarregar a conexão.

▸ **Hunt** (lin.fsid.cvut.cz/~kra/index.html) não é apenas um sniffer, mas uma ferramenta completa de exploração de falhas na pilha TCP/IP. Suporta tudo o que um sniffer deve ser capaz de oferecer: vigiar diversos tipos de protocolos (ICMP, TCP, ARP), montar mensagens fragmentadas a partir de pacotes TCP seqüenciais, detectar ACK Storms, seqüestrar sessões (hijacking) e "aprender" os MACs da rede, entre muitos outros truques. Por ser muito didático, é o preferido dos professores de cursos de redes e segurança – foi, inclusive, desenvolvido por um professor de matemática da Universidade de Praga, Pavel Krauz. Com o Hunt, podemos facilmente colocar-nos como man in the middle numa conexão, registrar tudo o que estiver sendo transmitido, manipular os dados e até mesmo terminá-la (reset).

▸ **ettercap** (ettercap.sourceforge.net) é um excelente sniffer ativo, ou seja, é especial para ser usado em redes com switches. Suporta MAC Flood e ARP Spoofing e é extremamente fácil de usar. Como funciona em modo texto, fica facilmente operacional em qualquer máquina Unix e é extremamente pequeno. Atacar com o ettercap é brincadeira de criança: primeiro, escolhe-se a dupla de máquinas que se quer monitorar as comunicações.



Escolhe-se, entre as conexões TCP ou UDP estabelecidas (pode haver mais de uma), qual é a que se quer monitorar. Depois disso, a tela mostra os dados que trafegam entre as duas máquinas. Pode-se gravá-los, examiná-los em modo ASC-II ou hexadecimal ou injetar car



acteres na trasmissão (e assim manipular a conexão). Há plugins e novos métodos de sniffing com nomes sugestivos, como Port Stealing, Hunt, Confusion... As possibilidades são enormes!

▸ **Angst** (angst.sourceforge.net), desenvolvido e testado para o OpenBSD, é uma ferramenta que tem a seu favor a robustez e o tamanho diminuto – além de fazer sniffing ativo. Possui menos recursos que seus primos mais ilustres como o ettercap e o dsniff, mas é pequeno o bastante para passar despercebido em uma máquina invadida. Funciona também com FreeBSD e Slackware Linux, outras plataformas Unix requerem recompilação e testes.

▸ **Dsniff** (www.monkey.org/~dugsong/dsniff) é, na atualidade, o mais respeitado sniffer ativo para redes comutadas. Foi testado pelos autores em OpenBSD e Linux em PCs e Solaris em máquinas SunSPARC, mas já foi testado em praticamente todos os sabores Unix conhecidos, entre eles AIX, FreeBSD, HP-UX e até Mac OS X (blafasel.org/~floh/ports/dsniff-2.3.osx.tgz). Há ainda uma versão (desatualizada mas funcional) para Windows, disponível em www.datanerds.net/~mike/dsniff.html.

Apesar dos sniffers mais simples serem mais fáceis de "plantar" e usar, há outros, mais elaborados, que são verdadeiras obras-de-arte do mal. O tcpdump sozinho em uma máquina Unix já poderia fazer muito estrago, e tem a vantagem de já estar instalado. Por outro lado, máquinas Win9x permitem que um invasor "esnife" a rede (usando, por exemplo, o Windump ou o Ethereal) sem invadir máquinas muito complicadas ou seguras. Como qualquer usuário de Win9x tem controle total sobre a máquina, é possível rodar um sniffer nela sem precisar quebrar nenhuma senha de administrador. A insegurança inerente a essa arquitetura auxilia o inimigo. Qualquer servidor ou estação, sejam eles Unices, Macintoshes, Netwares ou Windows NT/2k/XP, pode ser um perigo se mal configurados. Entretanto, redes com estações Windows 95/98/Me nunca serão seguras, qualquer que seja o esforço dispendido nelas.

Para mais opções, procure por "sniff" no Fresmeat (www.freshmeat.net) ou no Google (www.google.com). O download.com também possui diversas opções para Windows, e os CDs do livro também trazem mais alguns deles. Uma última dica sobre sniffers (e outras ferramentas): pedram.redhive.com.

Farejando na prática

Nada como uma sessão de sniffing para fixar os conceitos aprendidos. Poderíamos exemplificar esse procedimento com um sniffer gráfico como o Ethereal, que possui uma versão para Windows e uma bela interface de comando. O Ethereal ainda separa e interpreta os protocolos para você, de modo a fazer o máximo possível do "trabalho sujo" e deixar ao administrador de rede apenas com as informações úteis, "mastigadas". Mas lembre-se: se você está explorando uma rede remota, possivelmente vai deixar seu sniffer rodando sozinho para depois recuperar o arquivo contendo o tráfego. A interface gráfica, os "frufrus" e a inteligência do Ethereal de nada adiantarão nesse ambiente.

Antes de começar nosso estudo, dois lembretes. Em primeiro lugar, observe que sua conexão doméstica à Internet provavelmente se dá por um protocolo de camada 2 chamado de PPP ou Point-to-Point Protocol. Como o nome já diz, é um protocolo ponto a ponto: sua rede local possui apenas duas interfaces, a de sua máquina e a do modem do provedor de acesso. Justamente devido ao PPP, você não poderá farejar o tráfego na subrede de Internet à qual você está (aparentemente) conectado. Em uma ligação empresarial – normalmente por linha privada usando Frame Relay ou X.25 e não PPP – é possível (embora não muito provável devido a firewalls e roteadores) que a subrede da Internet determinada para a empresa possa ser farejada. De qualquer forma, um sniffer é útil apenas em estruturas de conectividade internas, onde há verdadeiramente várias máquinas trafegando informações na mesma rede local.

O outro lembrete não é menos importante. Existem três condições básicas para se poder "esnifar" a rede:

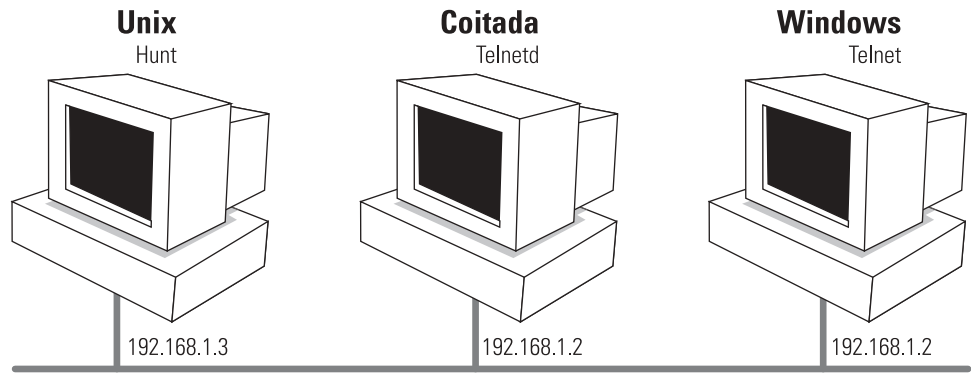
1. Existir uma rede com pelo menos 3 máquinas (as duas que se quer monitorar mais a sua, que monitorará as outras duas).

2. Possuir acesso de superusuário (root para Unix e Mac OS X ou Administrador para WinNT/2k/XP – máquinas com Win9x não precisam de nada disso...) na máquina que ficará em modo promíscuo e farejará a rede.

3. Possuir autorização para fazer a monitoração e alertar os usuários da rede para tal. Não é condição impeditiva para iniciar o sniffing, mas é ético. Todos têm direito, garantido por lei, à privacidade. Se sua empresa possui uma política de monitoração de tráfego, esta deve ser informada aos funcionários.

Isso posto, vamos ao que interessa. Em nosso primeiro estudo, usaremos o Hunt. O programa possui uma interface com o usuário em modo texto que, apesar de singela, é muito funcional. Não vamos entrar em detalhes sobre a instalação do programa – há intruções para isso na documentação que o acompanha.

Em nossa rede de testes (Laboratório de Redes II), instale o Hunt na máquina Unix. Ela será nossa máquina "haxor". Vamos monitorar uma conexão entre a máquina Windows e a máquina "coitada". Para que possamos acompanhar melhor a conexão, vamos abrir uma sessão de Telnet, que é um protocolo interativo e que funciona em texto simples – perfeita para ser monitorada pela sua simplicidade e baixa velocidade. Observe:



Na Coitada (192.168.1.2) estamos rodando um servidor de Telnet (não esqueça de ativá-lo!). Na máquina Unix, execute o Hunt. Será mostrada uma tela como esta:

```
/*
 *      hunt 1.5
 *      multipurpose connection intruder / sniffer for Linux
 *      (c) 1998-2000 by kra
 */
starting hunt
— Main Menu — rcvpkt 0, free/alloc 63/64 ——
l/w/r) list/watch/reset connections
u)     host up tests
a)     arp/simple hijack (avoids ack storm if arp used)
s)     simple hijack
d)     daemons rst/arp/sniff/mac
o)     options
x)     exit
->
```

O símbolo -> é o prompt do Hunt. Observe o menu. Há rotinas para listar, observar e derrubar conexões, verificar servidores online, faze hijacking etc. Temos que configurar o Hunt para que passe a escutar nossa rede. Observe: as opções l/w/r, respectivamente, listam todas as conexões ativas, vigiam uma delas e interrompem-na (reset). Para que funcionem, é necessário que haja alguma conexão ativa.

Escolha a opção "o" (options) e pressione a tecla Enter. Um novo menu será apresentado:

```
-> o
— options — rcvpkt 723, free/alloc 63/64 ——
l) list add conn policy
a/m/d) add/mod/del conn policy entry
c) conn list properties    mac n, seq n
g) suggest mac base        EA:1A:DE:AD:BE:00
h) host resolving              n       t) arp req spoof through req   y
```

```
r) reset ACK storm timeout     4s      w) switched environment        y
s) simple hijack cmd timeout   2s      y) arp spoof with my mac       n
q) arp req/rep packets         2       e) learn MAC from IP traffic   n
p) number of lines per page    0       v) verbose                     n
i) print cntrl chars           y
x) return
-opt>
```

São inúmeras opções. Por enquanto, vamos nos ater às mais básicas. As opções a/m/d manipulam o que é chamado, numa tradução livre, de regras de conexão. Basicamente, uma regra de conexão é a maneira pela qual dizemos ao Hunt quais conexões queremos monitorar. Vamos adicionar uma regra (opção "a").

```
-opt> a
src ip addr/mask ports [0.0.0.0/0]>
```

O programa pede o endereço IP e a máscara de rede da interface de origem (src) que queremos monitorar. No nosso caso, a interface de origem é aquela que vai fazer conexão com um servidor telnet, portanto colocaremos aí o IP da máquina Windows. Lembrando do que aprendemos em Redes II, 0 (zero) quer dizer "todos". Portanto, colocar um zero em qualquer lugar do endereço indica que todas as máquinas com aquele prefixo serão monitoradas. Por exemplo, se eu colocasse 192.168.1.1/32, eu monitoraria os pacotes originados nessa máquina. Se eu colocasse, por outro lado, 192.168.1.0/24, o Hunt farejaria os pacotes de todas as máquinas da rede 192.168.1.0, ou seja, entre 192.168.1.1 e 192.168.1.254. Num caso extremo, colocar 0.0.0.0/0 seria o mesmo que dizer ao programa: "Vasculhe TUDO!"

De maneira similar, preencha o endereço de destino. A próxima pergunta (insert at) é simplesmente para definir em que posição da lista nossa regra aparecerá.

A opção "l" lista as regras de conexão existentes. Em nosso caso, teríamos:

```
-opt> l
 0) 0.0.0.0/0 [all]                <-> 0.0.0.0/0 [23 513]
 1) 192.168.1.1/32 [all]           <-> 192.168.1.2/32 [all]
-opções do menu-
*opt>
```

Observe que temos um asterisco no prompt. O Hunt está nos indicando que as máquinas estão "vivas" na rede. Para sair do modo opções, usa a opção "x". De volta à tela inicial, temos a opção de listar ("l") as conexões ativas. Experimente e veja: não há nenhuma conexão no momento. Vamos criar uma então.

Na estação Windows (192.168.1.1) vamos fazer uma conexão de Telnet para a "Coitada". Abra uma janela do DOS e digite **telnet 192.168.1.2**. Se tudo estiver certo, vai aparecer a tela de login da máquina "Coitada" na janela de Telnet da máquina Windows. Na máquina Unix, volte ao Hunt e escolha a opção "l" novamente. Mágica: apareceu a conexão entre 192.168.1.1 e 192.168.1.2. A porta de origem é alta (como deveria ser, leia o capítulo Redes II e a RFC1700), e a de destino é a 23 – a porta do serviço Telnet. Vamos, então, pedir ao Hunt que mostre o tráfego para nós. Escolha a opção "w" (watch), escolha qual conexão quer monitorar e, em seguida, a opção "b" (both).

Volte à máquina Windows e dê seu login e senha. Se você observar no Hunt, essas informações serão mostradas lá. Caso o login seja bem sucedido, o promp do shell Unix aparecerá no Telnet da máquina Windows – e também na tela do Hunt, que está numa máquina que nem participou da transação! Faça várias experiências: liste diretórios, edite textos com seu editor favorito ou chame algum programa – todas essas atividades serão monitoradas pelo Hunt.

Para o Hunt, esse é o básico. Por sua simplicidade, acaba ficando limitado em possibilidades, mas é muito útil na maioria dos casos. É óbvio que há muito mais a explorar sobre o programa, mas deixo isso como dever de casa. Brinque com o Hunt por alguns dias, fareje sua rede, leia toda a documentação e procure por mais recursos na Internet. Depois, tente também brincar com o Snort, o Sniffit, o Ethereal e o IPTraf, tanto em Unix como em Windows. Use também algumas das ferramentas simples disponíveis nos CDs. Parar uma semana ou duas para tal seria o desejável, e só depois disso prossiga a leitura. Valerá a pena – e será divertido!

Quando a caça é vã

O Hunt é um ótimo (e didático...) sniffer, mas tem um problema insolúvel para os modelos clássicos de sniffers: não ustrapassa sniffers ou switches. Para as redes comutadas, é necessário utilizar sniffers apropriados, que façam ARP Spoofing ou, pelo menos, MAC Flood.

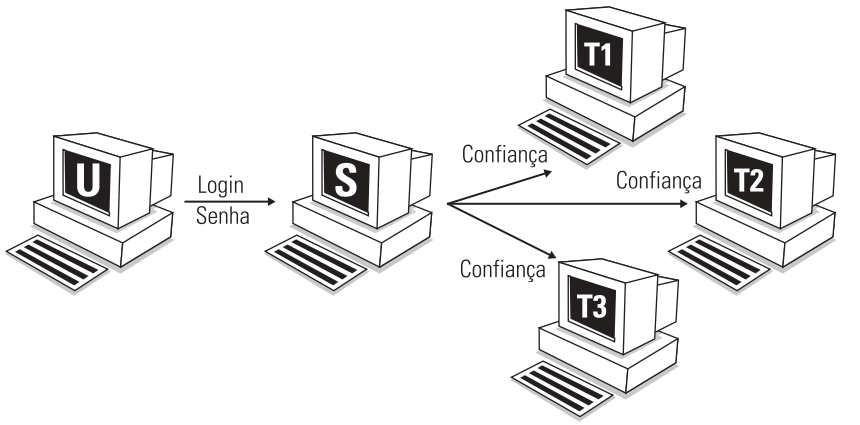
Dois sniffers o fazem de forma magistral: Ettercap e Dsniff. Já falamos sobre ARP Spoofing e MAC Flood há algumas páginas. Estude a documentação e experimente ambos os programas. Adicionalmente, sugerimos que o leitor procure mais informações sobre os seguintes métodos de sniffing:

- SSL Spoofing (falseando ou manipulando certificados digitais);
- DNS Spoofing (desviando o tráfego Web para servidores falsos);
- Snarfing (uma maneira de "pentear" os pacotes TCP/UDP para extrair deles apenas a informação que é mostrada na tela da vítima: e-mail, mensagens instantâneas, páginas da web...).

Who can you trust?

Just say enough is enough / Oh I gotcha / Who who who can you trust[7]

Uma maneira de se conseguir um Spoofing mais efetivo é através das relações de confiança (ou trust relations) entre computadores. Isso permite que um computador possa ter acesso a vários recursos fazendo login em apenas um sistema.



Observe: existe uma máquina servidora (vamos chamá-la de "S") que possui relação de confiança com outras máquinas (vamos chamá-las "T1", "T2", "T3" etc.). Se o usuário logar-se na máquina S, automaticamente terá acesso a recursos das máquinas "T". Enquanto essa facilidade "quebra um galhão" para administradores de rede e permite que usuários loguem-se em apenas um sistema para ter acesso a vários, se um hacker conseguir fazer-se passar pela máquina que tem a confiança poderá ter acesso a todas as outras, que "confiam" nela.

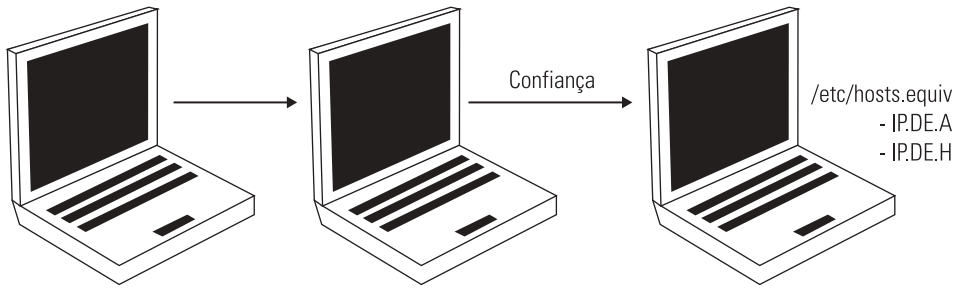
Tanto sistemas Unix quanto Windows (e também Novell...) possuem facilidades parecidas. O Windows 2000 possui um recurso chamado Advanced Directory Service (ou ADS), que detém o cadastro de todos os usuários e máquinas da rede e as relações entre eles. O usuário, ao logar-se em um domínio Windows, tem seu nome e senha comparados com os do ADS e, caso aceitos, todos os serviços são disponibilizados.

A Novell possui um produto chamado eDirectory (www.novell.com/pt-br/products/edirectory) baseado em um núcleo, o Novell Directory Service ou NDS. O software opera da mesma forma que o ADS da Microsoft, com um diferencial: não está restrito aos produtos Novell. Na realidade é possível, através do eDirectory, criar conexões para todos os tipos de plataformas de software e hardware imagináveis, incluindo aí Unices, Windows e mesmo sistemas de grande porte.

Tanto o eDirecory quanto o ADS baseiam-se num padrão chamado Lightwheight Directory, Access Protocol ou LDAP. Há versões do LDAP disponíveis para diversos sabores de Unix. Uma das mais famosas é o OpenLDAP (www.openldap.org), versão livre, gratuita e de código aberto. O LDAP é uma forma segura de prover, para aplicações Unix (e também Windows e Macintosh...), o chamado single-sign-on (ou log-se apenas uma vez) e ter acesso a vários recursos disponíveis em pontos distantes da rede.

Mas os sistemas Unix possuem dois antepassados desses serviços: o NIS (Network Information Service) e o famigerado Unix Trust. Por sua idade, ambos são extremamente inseguros e facilmente hackeáveis. Falaremos do NIS mais para frente, já que sua falha não se trata de IP Spoofing.

O Unix possui uma série de comandos, todos iniciando em "r" (de remote) que permitem que diversos sistemas possam ser operados sem que seja preciso autenticar-se em cada um deles. Imagine, por exemplo, que o usuário está logado na máquina A. A máquina B confia em A, portanto permitirá que os comandos "r" sejam executados nela também. Por exemplo, se o usuário da máquina A emitir o comando rlogin IP.DE.B, será presenteado com um shell da máquina B sem que seja necessário fornecer usuário ou senha. B confia em A, então B confia nos usuários de A... Da mesma forma, há os comandos rsh (remote shell – permite a execução de apenas um comando), rcp (remote copy), rmail (lê e-mails no outro sistema) entre outros. Para que seja permitida a execução dos comandos, é necessário que o IP de A esteja contido no arquivo /etc/hosts.equiv de B. Todas as máquinas cujo IP estão em /etc/hosts.equiv de B são de confiança para B. Além desse arquivo, cada usuário de B pode ter, em seu próprio /home, uma lista de máquinas nas quais confia, guardada no arquivo .rhosts.



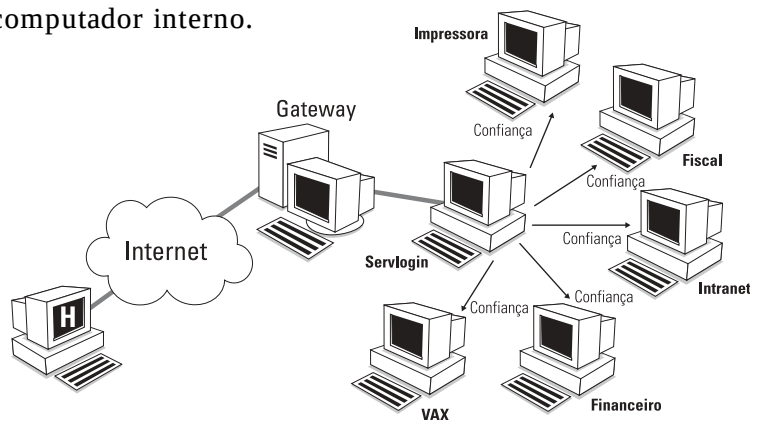
No exemplo acima, todas as máquinas "S" e "T" são Unix. As estações podem ser qualquer coisa. Em cada uma das máquinas "T" mostradas, há um arquivo, o /etc/hosts.equiv, contendo o IP de "S". Assim, todos os usuários que se logarem em "S" poderão ter acesso aos recursos das máquinas "T". É possível crial shell scripts com os r-comandos, por exemplo, para automatizar o processo e facilitar a vida desses usuários.

7. "Apenas diga que já teve o bastante / Hehe, eu pegei você / Em quem, em quem, em quem você pode confiar?" Joan Jett, Who can you trust. Do album "Cherry Bomb, de 1995.

Mas o que acontece quando usamos os r-comandos? O usuário está logado em "S" e, por meio de um comando rlogin, conseguiu um shell de "T3" sem que fossem solicitados um usuário e uma senha válidos em "T3". Ocorre que, na prática, o usuário está logado em "T3", e todas as suas ações serão originadas de "T3" (e não de "S" ou de "H", sua estação de trabalho...). Se o invasor usar algum scanner contra o site de uma empresa, ou tentar conectar-se por brute-force a um serviço na Internet, será o IP de "T3" que será registrado, não o de "S" e muito menos o de "H".

Agora é que a mágica acontece. Se formos usuários já cadastrados em "S", não há problema. Basta logarmos em "S", dar um rlogin para T3 e usá-lo. Mas e se não formos? E mesmo se formos, seremos registrados nos logs de "S", então o que fazer?

É possível "envenenar" uma relação de confiança entre duas máquinas usando spoofing, um pouco de sniffing e uma pitada de negação de serviço. Imagine uma rede corporativa onde há uma máquina chamada SERVLOGIN, na qual os usuários serão autenticados, e outras, cada uma com um nome alusivo ao recurso que ela disponibiliza (por exemplo, IMPRESSORA, FISCAL, INTRANET, VAX, FINANCEIRO, entre outros...). Todas essas máquinas usam o /etc/hosts.equiv para "confiar" no julgamento de SERVLOGIN no tocante a quais usuários têm permissão para acessar seus recursos. Imagine também que, em algum lugar ignorado na Internet, há uma máquina "H" que deseja conectar-se a algum computador interno.



Para realizar a tarefa a contento, temos que partir de algumas premissas:

- Em primeiro lugar, temos que ter em mente que "H" não está na mesma LAN que SERVLOGIN ou qualquer uma das outras máquinas. Pelo contrário, está lá – beeeem longe – na nuvem da Internet. Por isso mesmo, é impossível "esnifar" o que ocorre na LAN.
- De alguma maneira já descobrimos que SERVLOGIN é considerado "confiável" pelos outros servidores. Descobrimos isso porque invadimos IMPRESSORA, por exemplo, ou o próprio SERVLOGIN, e consultamos os arquivos .rhosts ou /etc/hosts.equiv ou mesmo os logs do sistema, à procura de sinais de conexão por Unix Trust.
- Nosso objetivo é, então, traçado: queremos que uma das máquinas (por exemplo, INTRANET) "pense" que "H" é, na realidade, SERVLOGIN. Com isso, teremos um shell em INTRANET, com um IP à disposição para usarmos à vontade em nossas maldades por aí...

Vamos dar uma "receitinha de bolo" para esse ataque. É a melhor maneira de explicar a teoria de cada um dos passos necessários. Para maior clareza no texto, chamaremos cada um dos computadores apenas por seus nomes (i.e. "H" em vez de "a máquina H" ou "S" em vez de "o servidor S").

1. "H" inicia várias – na verdade, milhares de – conexões reais, sem impostura (ou seja, sem spoofing), para INTRANET. "H" envia vários pacotes SYN e espera pelos ACKs de INTRANET. Baseado nesses ACKs, "H" pode inferir (ou antes, adivinhar) a progressão de números seqüenciais TCP gerados por INTRANET. Com isso, "H" pode ter uma idéia dos números a usar mais para frente, quando estiver conectando ao sistema INTRANET.

2. Depois disso (ou simultaneamente, caso o invasor possua outra máquina) lança-se um ataque de negação de serviço contra SERVLOGIN. Com SERVLOGIN fora da jogada, impedimos que o mesmo envie um pacote TCP RST e derrube nossa conexão "spoofada".

3. Usando um dos r-comandos, "H" inicia uma conexão a INTRANET usando o IP de SERVLOGIN. A INTRANET responde com um ACK para SERVLOGIN, que está fora de combate devido à negação de serviço.

4. Agora, a mágica: "H" envia um ACK para INTRANET, com o IP de SERVLOGIN e uma estimativa do número seqüencial TCP – calculado pela progressão detectada no passo 1 mais o tempo que o processo todo levou até chegar aqui.

5. Caso tenha acertado na mosca (o hacker possui apenas um tiro...) a comunicação é estabelecida e mantida enquanto SERVLOGIN estiver fora do ar. Se tiver errado, o invasor pode repetir a receitinha até acertar.

Alguns sistemas são idiotamente fáceis de prever a seqüência TCP, outros nem tanto, e há uns poucos onde essa seqüência (?) é quase aleatória (?!?!??). A predictabilidade dela dirá se o sistema é facilmente hackeável por esse método – ou não.

Lembra daquelas velhas cantigas do tipo "João amava Maria que amava Pedro..."? Pois é. No passo 3, "H" iniciou uma conexão com INTRANET fingindo ser SERVLOGIN. Para iniciar essa conexão, usou um r-comando (por exemplo, um rlogin). Se o hacker acertou a seqüencia TCP, foi presenteado com um shell de INTRANET. Só que INTRANET "pensa" que quem iniciou a comunicação é SERVLOGIN, e manda as respostas para essa máquina. Resultado: "H" pode emitir comandos, mas não tem a mínima idéia se estão funcionando ou não – não há feedback na tela.

Enquanto isso pode parecer suficiente para um estrago qualquer (assim como nas vulnerabilidades discutidas anteriormente, como o estouro de pilha ou os CGIs "amigos"), o hacker pode usar esse shell (e também o do estouro de pilha, bem como o dos CGIs...) para configurar rapidamente um backdoor em INTRANET. Não se esqueça que SERVLOGIN está tinto com a negação de serviço, mas recobrará os sentidos a qualquer momento e cortará a comunicação

(com um TCP RST) entre INTRANET e "H". Entre as coisas que o invasor pode fazer na máquina INTRANET estão:

- Colocar o IP de "H" no /etc/hosts.equiv;
- Criar um usuário com direitos de root no /etc/passwd;
- Implantar qualquer tipo de backdoor.

Há mais opções além dessas. A partir dessas alterações, o hacker pode instalar vários backdoors, incluir mais máquinas no hosts.equiv (ou mesmo o símbolo + +, que faz com que INTRANET confie em qualquer um...) e criar outras contas no sistema. É claro que administradores atentos logo notarão tais mudanças, por isso é bom usar essa máquina para rapidamente "0wnar" outras. Não esqueça ainda que, por mais engenhoso que o procedimento seja, ferramentas de IDS e auditoria de arquivos logo descobrirão a façanha.

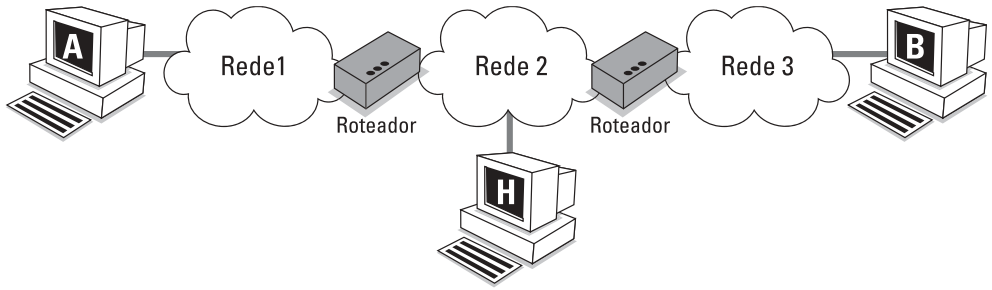
Como tarefa para casa, sugerimos ao leitor que procure documentação na Internet sobre os seguintes tipos de Spoofing:

- DNS Spoofing
- Source Routing
- Proxy Spoofing
- Daisy-chain Spoofing

Captura de sessões (Hijacking)

Lembro-me das aulas de matemática da primeira série do primário (hoje isso é chamado de Educação Fundamental, mas é tudo a mesma coisa...). Problemas matemáticos eram resolvidos dividindo a folha do caderno em três campos: sentença matemática, cálculo e resposta. Na sentença matemática, colocávamos o que fui, anos depois, aprender que se chama forma algébrica da equação (ou da inequação ou fórmula). A sentença matemática deveria ser algo claro e bem detalhado, para que a pessoa que fosse ler a resolução do problema pudesse entendê-lo num relance. No campo Cálculos, as coisas eram mais livres. Podíamos preencher completamente o espaço com cálculos e anotações – mantendo, obviamente, alguma "limpeza", senão era nota zero! Na resposta, nos era permitido colocar apenas o resultado de nossos cálculos. Em nossos estudos sobre invasões, podemos colocar, em nossa sentença matemática, o seguinte problema: Sniffing + Spoofing = ?. Façamos os cálculos, então, e vejamos no qual isso vai dar.

Nas páginas anteriores vimos que sniffing é um meio de saber o que se passa numa rede local – mesmo que os dados em trânsito não sejam de nossa conta. Por outro lado, podemos enviar dados a outros computadores fingindo ser uma terceira pessoa – isso é spoofing. É fácil perceber que, combinando essas duas técnicas, podemos "roubar" uma sessão entre dois computadores. "Como assim roubar?", o leitor perguntaria. Uma imagem vale por mil palavras:



Observe que A, B e H não precisam estar na mesma LAN. H pode estar, com efeito, na LAN de A, na de B ou em qualquer subrede intermediária. A malha de roteadores mostrada poderia ser substituída por uma "nuvem" representando a rede como um todo – poderia mesmo ser a Internet. O importante é notar que H tem de estar em algum ponto pelo qual estejam passando os pacotes entre A e B. A conexão entre A e B poderia ser qualquer coisa, como uma chamada HTTP ou correio eletrônico sendo transmitido via SMTP. Capturando uma sessão de conversação por ICQ, por exemplo, podemos continuar conversando com B enquanto ele pensa que somos A. Capturando uma sessão de FTP, SMB (redes Microsoft) ou Telnet entre o usuário A e o servidor B, por exemplo, podemos navegar pelos servidores sem precisar fazer login – A já o fez anteriormente, e B agora pensa que somos A. Acendeu uma "luzinha" aí? Pois é, há outras maneiras de conseguir acesso a sistemas além do brute force e buffer overflow...

Realmente, por mais que os mecanismos de autenticação sejam seguros (senhas de mão única, criptografia, assinaturas digitais, etc.), pode-se capturar uma comunicação qualquer DEPOIS que a autenticação foi feita e, assim, pular essa parte chata. As produtoras de software normalmente gastam milhões em desenvolvimento de esquemas seguros de autenticação e esquecem-se do que, uma vez feita, o sistema irá SEMPRE acreditar que o usuário é o mesmo – e, como vimos, nem sempre é...

Outra característica de uma sessão de hijacking é que não é preciso estimar ou adivinhar a progressão e a preditabilidade dos números seqüenciais TCP. O atacante, estando no meio do caminho entre as duas estações, tem acesso à progressão real dos números seqüenciais TCP e pode, portanto, controlá-los quando estiver fazendo se passar por outrem. Em nosso exemplo , H está cuidadosamente registrando toda a comunicação entre A e B, incluindo aí as seqüências TCP.

Na prática, os passos a serem seguidos são:

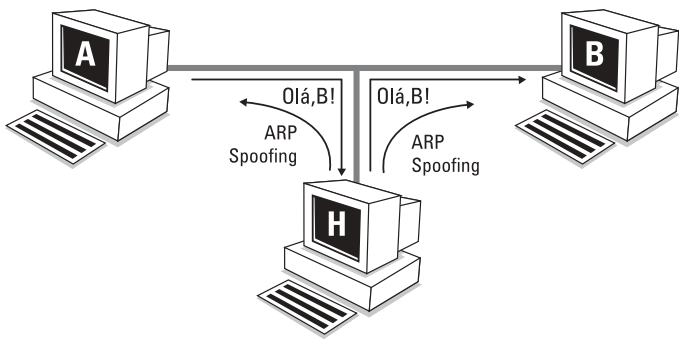
1. H deve observar – com técnicas de sniffing – a comunicação entre A e B por um tempo razoável, para determinar o tipo de conexão, o protocolo em uso (HTTP, SSH, FTP, Telnet, MSN, ICQ...) e a seqüência TCP nos dois sentidos. Tudo isso deve ser registrado em um arquivo e analisado. Quando a conexão estiver bem "escarafunchada", decide-se que é interessante seqüestrá-la e determina-se os meios para tal, aí podemos iniciar a captura.

2. H começa então – com técnicas de spoofing - a "enganar" B, fazendo-o pensar que H é A. A forma mais usual é H simplesmente criar tráfego entre ele

e B, gerando pacotes cujo endereço IP de origem seja o de A. B pensará que os pacotes vêm de A.

3. Uma vez que B aceita comandos de H pensando serem de A, pode-se controlar B a partir de H. Se for uma sessão de Telnet ou de SSH, H pode operar o computador B por comandos shell. Se for uma sessão FTP, H pode baixar arquivos de B que estejam liberados para A – e bloqueados para usuários de FTP anônimo. Se for uma conversação MSN, H pode continuar a conversa com B fingindo ser A.

Olhando "por cima", parece que funciona. Mas há um problema: A ainda está viva e B devolve a ela pacotes TCP com o bit ACK ligado cada vez que H injeta tráfego na linha. A tentará ressincronizar a conexão e responderá a B na mesma medida. Além desses ACKs espúrios gerados pelos pacotes injetados por H na conexão, há ainda os pacotes normais da conexão entre A e B. Isso causará o que chamamos de tempestade ACK ou *ACK storm*.



Quando descrevemos, páginas atrás, um spoofing baseado em Unix Trusts, retiramos do ar a máquina SERVLOGIN (aquela pela qual queríamos nos fazer passar) por meio de um ataque do tipo Negação de Serviço. Naquela ocasião, o fizemos para evitar que SERVLOGIN derrubasse a conexão (que afinal não fora gerada por ela) com um pacote TCP RST. O caso aqui é ligeiramente diferente: A realmente iniciou a conexão entre ela e B. O tráfego injetado por H aproveita-se disso e a tempestade de ACKs é um efeito colateral indesejado, em vez de ser um procedimento normal do protocolo TCP, como no caso do RST.

Apesar de ser efetivo naquele caso específico – não precisávamos de SERVLOGIN no decorrer da ação –, usar Denial of Service nem sempre é desejável:

- A máquina fora do ar pode voltar a qualquer momento. Não há previsão de quando, e certamente o tempo que ficará desabilitada será muito pequeno – o suficiente para um ou dois comandos.
- Se o computador que sofreu o DoS for atendido por alguém, ou seja, houver usuários conectados nele – por exemplo, uma estação de trabalho –, o ataque será facilmente detectado.
- Em alguns casos, é necessário que o computador pelo qual queremos nos fazer passar fique "vivo" – há recursos que só podem ser destravados por ele, ou precisamos que o usuário acredite que ainda está conectado.
- Era imperativo tirar a máquina do ar devido aos TCP RST, que cancelam a conexão correspondente. Aqui não há RST pois o originador da conexão foi – realmente – A e não H. A passa a ser necessário e não indesejado.
- Negação de Serviço é algo deselegante – e você é vaidoso, não é?

A resposta está em algumas páginas um pouco mais para trás. Quando fizemos sniffing ativo, usamos a técnica de ARP Spoofing para confundir o switch e as próprias estações, colocando nossa máquina no meio da conexão. Para tal, usamos o programa Hunt em uma configuração man-in-the- middle. Bem, podemos usar a mesma idéia para evitar o ACK Storm. Com o tráfego passando todo pela máquina que está escutando a rede, podemos controlar a propagação desses ACKs espúrios.

Nessa configuração, fica claro que, como não há problema de tempestade de ACKs, podemos emitir comandos para a máquina destino (B, em nosso exemplo). Obviamente, quando a conexão entre H e B for encerrada, a diferença entre os TCP sequence numbers que A envia e que B espera é tão grande que a sincronização entre A e B é impossível – e a conexão entre eles também cai. Programas de Hijack modernos (como o Hunt e o Dsniff) possuem, entretanto, ferramentas para ressincronizar as conexões, assim uma desconexão gratuita não irá causar desconfianças.

Selecionando seus combatentes

É claro que há a possibilidade de se fazer tudo isso manualmente. Entretanto, a maneira mais fácil é usar ferramentas especialmente construídas para tal. Uma delas é o Hunt, visto na sessão sobre sniffing. O Hunt possui duas opções interessantes: ***simple hijack*** e ***arp/simple hijack***. No menu opções, ele oferece algumas possibilidades também muito interessantes, como ***arp spoof with my mac*** (lembram-se do ARP Spoofing?), ***learn MAC from IP traffic*** e ***switched environment*** (ferramentas para enganar bridges e switches).

O Dsniff, outra suíte de sniffing comentada anteriormente, também possui ferramentas para captura de sessões. Além do próprio programa Dsniff (especializado em sniffing), a suíte possui as seguintes ferramentas:

Ferramentas de sniffing incluídas na suíte Dsniff:

- **filesnarf**: copia na máquina do hacker os arquivos trafegando em uma conexão NFS (Network File System) entre dois outros computadores.
- **mailsnarf**: reproduz na máquina invasora mensagens de e-mail sendo transmitidas por POP ou SMTP. Com modificações no código do programa, é possível ler também mensagens IMAP e UUCP. As mensagens são armazenadas em formato mailbox – legível por praticamente todos os programas de e-mail existentes.
- **msgsnarf**: registra toda a conversação entre duas pessoas que estejam usando os serviços de mensagem instantânea AOL Instant Messenger, ICQ 2000, IRC, MSN e Yahoo.

- **urlsnarf**: fareja requisições HTTP e as apresenta no formato Common Log Format ou CLF.

Ferramentas de spoofing incluídas na suíte Dsniff:

- **arpspoof**: ferramenta para ARP Spoofing.
- **dnsspoof**: forja respostas a solicitações DNS em uma LAN. É útil para contornar regras de acesso baseadas no nome do host ou para implementar diversos ataques tipo man-in-the-middle baseados em DNS.
- **macof**: uma ferramenta para MAC Flooding. Inunda a LAN com uma multidão de endereços MAC randômicos, fazendo com que bridges e switches vulneráveis passem a se comportar como hubs – ou seja, deixando passar todo o tráfego indiscriminadamente por todas as portas.

Ferramentas de captura e controle (hijacking) incluídas na suíte Dsniff:

- **tcpkill**: derruba a conexão selecionada.
- **tcpnice**: controla a velocidade da conexão entre dois nós sem prejudicar o restante da rede. É interessante para reduzir a velocidade de uma conexão e, com isso, monitorá-la "ao vivo".
- **sshmitm**: age como um intermediário (uma espécie de proxy) para conexões SSH. Uma vez desviada a conexão (com ferramentas como, por exemplo, o dnsspoof), o sshmitm pode farejar o tráfego à procura de logins e senhas e mesmo capturar a sessão.A porção **mitm** do nome significa man-in-the-middle.
- **webmitm**: age também como intermediário, desta vez para conexões HTTP/HTTPS (sim, suporta SSL!!!). Útil para conseguir senhas de acesso a sites e informações normalmente inseridas em formulários como números de cartões de crédito e informações "criptografadas".

Além do Hunt e do Dsniff, há algumas outras ferramentas para seqüestro de sessões. Uma delas, muito conhecida, é o Juggenaut (packetstorm.linuxsecurity .com/new-exploits/1.2.tar.gz). Outro também muito comentado é o IP-Watcher, produto comercial (e pago) da Engarde Systems – www.engarde.com) . Um detalhe: todos eles são para Unix. Não há boas opções de ferramentas para seqüestro de sessões que rodem em Windows ou outras plataformas – neste caso específico, meu amigo, você está preso aos Unix. Uma saída é dotar seu Windows de um ambiente Unix simulado como o Cygwin (www.cygwin.com) e rodar essas ferramentas lá. Para isso, será preciso instalar também as bibliotecas pertinentes a cada um dos softwares dentro do Cygwin. Outra maneira é dotar seu Windows de uma máquina virtual PC completa, como o comercial e caro VMWare (www.vmware.com) ou o livre e gratuito Bochs (bochs.sourceforge.net), e rodar algum Unix para PC (Linux, FreeBSD, Solaris) dentro dele.

(ou: como me tornei membro do Al Qaeda)

Anteriormente descrevemos uma sessão de sniffing com o Hunt. Chegamos ao ponto de verificar os comandos que A emitia para B e ver as respostas que B enviava para A. Ainda não estávamos exatamente no meio da conversação – simplesmente farejávamos o tráfego ("pô, passam por aqui berrando e não querem que eu escute...").

Vamos refazê-la, desta vez capturando a sessão. Usaremos nossa rede de testes e as mesmas instalações que usamos em nosso experimento anterior. Usaremos também a mesma sessão de Telnet que descrevemos.

Rapidamente relembrando o procedimento já visto, selecione a opção "o" (options) e adicione uma *conn policy*, de acordo com os IPs das máquinas cujas conexões queremos monitorar e, posteriormente, capturar. Digite "l" para ver se há interfaces "vivas" que obedeçam a essa regra (observe o prompt do Hunt). Depois disso, digite "x" para voltar à tela principal e "l" novamente para listar as conexões existentes que estejam de acordo com as *conn policies* definidas. Selecione "w" (watch) para monitorar uma das conexões, e depois escolha a conexão de nossa sessão telnet (lembre-se: estamos fazendo um telnet da máquina Windows para a Coitada). Logue-se por Telnet e emita alguns comandos. Se tudo estiver certo, todos os dados em ambos os sentidos da conexão serão ecoados no Hunt – que está rodando na estação Unix. Até aqui era o que havíamos feito.

Note que, se você estiver em um ambiente de rede segmentado (por exemplo, há um switch em sua LAN), você terá de fazer um ARP Spoofing para enganá-lo – caso contrário, não conseguirá ver as conexões que estejam em outro segmento. O próprio Hunt pode fazer isso. Selecione "d" (daemons) e "a" (arp spoof + arp relayer daemon), e configure os endereços IP de origem e destino pelos quais você quer fazer ARP Spoofing (opção "a" – add host to host arp spoof). Digite "s" para iniciar o daemon e espere alguns minutos. Dependendo do tráfego de sua rede e das estações em questão, pode demorar de um a 20 minutos para que o switch passe a distribuir pacotes indiscriminadamente.

Com ou sem switch, você está pronto para seu primeiro seqüestro. Volte ao menu principal e escolha a opção "a" (arp/simple hijack). Será perguntado qual das conexões deseja capturar. Selecione a conexão de nosso telnet e responda às perguntas sobre conexão e apresentação dos dados. A tela de captura é, em um primeiro momento, idêntica à tela de observação (watch). O comportamento também é o mesmo, por enquanto: tudo o que o usuário da máquina Windows fizer será ecoado para o Hunt antes de ser enviado à "Coitada", o mesmo ocorrendo com a resposta de "Coitada" para Windows.

Isso ocorrerá indefinidamente, até que o *Hacker* deseje capturar a conexão. Para tal, basta pressionar Control+C. Na tela, aparecerá a mensagem:

```
— press any key>
```

Assim que uma tecla qualquer for pressionada, a tela do Hunt se parecerá com isso:

```
– press any key>
you took over the connection
CTRL-] to break
```

Pronto! Já estamos em modo interativo com a sessão capturada. A máquina Windows não tem mais controle sobre a conexão, tudo o que o usuário digitar será ecoado no Hunt na cor verde, mas não será enviado à Coitada. Por outro lado, tudo o que o *Hacker* digitar no Hunt será enviado à máquina Coitada. Como é uma sessão de Telnet, digitar comandos de shell no Hunt resultará em sua execução em Coitada.

```
– press any key>
you took over the connection
CTRL-] to break
ls
ls              Comandos digitados pelo
ls              usuário que perdeu a conexão
exit
cazzo!


coitada [/home/usuario] > ls
Desktop  README  Xresources  Xsetup  aliases.sh  chooser  j  tmp


coitada [/home/usuario] > w
  1:44am  up 7 days,  5:44,  4 users,  load average: 0.99, 1.22, 1.20
USER     TTY      FROM           LOGIN@   IDLE   JCPU   PCPU  WHAT
root     tty1     -              Wed 3pm  3:48m  0.07s  0.07s  -bash
usuario  pts/2    192.168.1.1     1:07am   0.00s  0.03s  0.01s  w
root     pts/0    -               29May03  7days  0.00s   ?     -
root     pts/1    -              Wed 3pm  9:28m  0.15s  0.15s  /bin/bash
coitada [/home/usuario] >
```

Na tela da máquina Windows será mostrada a seguinte informação:

```
$ls
$ls
$ls
$exit
$cazzo!
```

Ou seja, o usuário tem uma realimentação do que digitou, e nenhuma mensagem de falha será mostrada – simplesmente não acontece nada! Essa saída é, na verdade, fornecida pelo Hunt. Ele inclui o caráter $ para dar uma impressão de prompt do shell – *lie and deceive...*

Com controle sobre "Coitada", o *hacker* pode tentar o que quiser: colocar um backdoor, apagar tudo, fazer download de documentos e arquivos importantes (inclusive o arquivo de senhas), usar a máquina como trampolim para outro ataque (mais uma maneira de spoofing...), tentar ganhar acesso irrestrito, etc, etc, etc. É muito importante observar que usamos um servidor Linux como exemplo, mas o procedimento é o mesmo para qualquer plataforma. Poderíamos tranqüilamente ter usado o Hunt para seqüestrar uma sessão entre uma estação Windows 98 e um servidor Windows 2000 com IIS e Telnet habilitado (ou FTP ou HTTP ou POP...).

Quando cansar de brincar, o invasor pressiona Control+] para sair. Ele terá, ainda, a opção de derrubar a conexão (e o usuário na máquina Windows receberá uma falsa mensagem de problemas com a rede) ou ressincronizá-la. O Hunt é inteligente nessa hora: ele tem registrados os números TCP seqüenciais vindos da máquina Windows e da própria máquina Unix, na qual está o programa. Dessa forma, o Hunt sabe quantos pacotes Windows tem de enviar para o "limbo" antes de permitir sua reconexão à máquina Coitada. Por exemplo, se o Hunt enviou 45 pacotes TCP para a Coitada, quando a captura da sessão for finalizada a mensagem

```
msg from root: power failure – try to type 45 characters
```

será mostrada na tela de Windows. Assim que o usuário digitar a 45ª tecla, os números seqüenciais TCP estarão novamente em sincronia. A sessão pode continuar normalmente – e, na maioria dos casos, o usuário em questão sequer perceberá o que aconteceu. Mas tenha em mente que a maioria das vezes não significa todas, e um usuário que conheça o comportamento do Hunt facilmente identificará a falcatrua e avisará a autoridade competente.

Tarefa para casa: tente fazer exatamente o mesmo com o Dsniff (**leia a documentação!**). Você verá que, por não ser uma ferramenta integrada como o Hunt, mas um conjunto de pequenas ferramentas de uso específico, terá de usar várias delas para obter o mesmo efeito. Entretanto, no mais perfeito estilo Unix de fazer as coisas, também verá que é bem mais fácil automatizar o ataque com um shell script que faça uso delas.

Outros métodos de desviar tráfego

Há muitas outras formas de colocar-se na posição de man-in-the-middle e fazer o tráfego passar por seu computador antes de chegar ao destino. Algumas delas incluem desviar tráfego web por meio de DNS Spoofing. Há ferramentas (como o dnsspoof, incluso na suíte Dsniff) que podem ser programadas para enviar falsas respostas a requisições DNS. Por exemplo, o site da Digerati está hospedado no host 200.246.179.102. É possível usar o dnsspoof para fazer um internauta qualquer acreditar que, em vez disso, o domínio digerati.com.br está em 200.230.xxx.yyy. Lá pode haver um site falso ou um sniffer que grave as informações passadas e redirecione o tráfego ao site verdadeiro.

Outra forma de desviar o tráfego bastante empregada faz uso do Netcat – uma ferramenta poderosíssima e presente em todos os Unix e no Windows.

Veremos os diversos usos do Netcat no próximo capítulo.

Uma última dica: há um excelente documento em packetstorm.linuxsecurity.com/new-exploits/ssh-insertion-attack.txt que discorre sobre captura de sessões usando o protocolo Secure Shell ou SSH. O SSH é, grosso modo, um primo do Telnet cuja conexão é toda criptografada. Bem, esse documento ensina a "meter-se" nessa conexão que muitos consideram segura.

Negação de serviço (Denial of Service)

Nem sempre queremos obter acesso a algum sistema. Muitas vezes, por vingança, terrorismo ou simples depredação, queremos apenas derrubar um site, sistema ou rede e causar o máximo possível de prejuízos à vítima. Tal procedimento é chamado atualmente de *Negação de Serviço* porque o resultado é, via de regra, a indisponibilidade temporária ou permanente do serviço que estamos atacando.

Outras vezes um Denial of Service (ou DoS, como é normalmente chamado) é necessário como parte de um ataque maior, como vimos em nossos procedimentos de spoofing neste mesmo capítulo. De uma forma ou de outra, um ataque do tipo DoS é o cibercrime mais deselegante que se pode cometer – é como "partir para a ignorância", como diziam os antigos. Sem entrar em detalhes sobre as conseqüências mercadológicas, políticas e financeiras de um ataque DoS bem-sucedido, podemos considerá-lo tão mortal quanto moralmente baixo, da mesma forma que uma briga de rua o é.

Para entender a negação de serviço, imagine qualquer sistema conectado à Internet que se queira derrubar como sendo um serviço "no mundo real"; floricultura, pizzaria, polícia, defesa civil, etc... Imagine, por exemplo, que você queira colocar fogo em um prédio público e ter certeza de que não terá seus planos arruinados por algum bombeiro com vocação para herói. Uma das maneiras de impedir que os bombeiros entrem em ação é impedir que eles saibam que o prédio está em chamas. Para tal, basta manter todas as linhas do 193 congestionadas com trotes. Emergências legítimas nunca serão atendidas.

Ainda no âmbito telefônico, um belo exemplo de negação de serviço pôde ser visto no filme *Duro de Matar 3*: o terrorista interpretado por Jeremy Irons colocou uma bomba em alguma escola primária do município de Nova York. Além de não informar à polícia em qual escola o aparato estava instalado, o bandido ainda divulgou a presença da bomba para a mídia. Resultado: a população inteira da cidade começou a ligar para os números da polícia, congestionando as linhas e impedindo os policiais de trabalhar.

Negação de serviço local

Há diversos níveis de DoS. Os mais básicos são os causados por pessoas com acesso físico ao sistema – em outras palavras, literalmente "meter a marreta" no equipamento. Passar com o caminhão por cima da lombada eletrônica para tirá-la de serviço (e livrar-se da multa) também pode ser considerado um tipo de negação de serviço.

Um DoS de nível um pouco mais alto (não mais na sarjeta, mas ainda mendigando na calçada) seria o acesso lógico aos sistemas. Usuários com contas ou invasores que conseguiram acesso limitado podem tentar destruir o sistema com comandos nocivos. Há diversas maneiras de fazer isso.

- **Apagamento ou destruição:** um usuário com uma conta em um servidor Solaris poderia tentar um **rm -Rf *** em um diretório sensível – o /etc, por exemplo. Caso o ambiente seja Windows, nada mais fácil que um FORMAT C: ou um DEL *.* /S/Q.

- **Consumo de recursos**: mesmo usuários com acesso muito restrito devem poder rodar programas no sistema – como trabalharão sem eles? Isso posto, é possível criar programas especialmente talhados para aumentar em progressão geométrica o consumo de recursos da máquina – seja por múltiplos acessos a disco, inundação das interfaces de rede com tráfego espúrio, multiplicação indefinida de processos ou mesmo gravação ininterrupta de dados em arquivos até preencher todo o espaço disponível.

- **Vulnerabilidades locais**: um buffer overflow, se bem-feito, pode levar a acesso irrestrito ou à execução de comandos arbitrários no sistema vulnerável. Um ataque mal-feito, entretanto, pode apenas travar o programa. Se o programa for vital para o sistema (como partes expostas do kernel em um Windows NT) ou oferecer um serviço aos usuários externos (como o X-Window ou o Sendmail em um Unix), o estrago será maior. Caso o lixo jogado na pilha seja em quantidade realmente grande, é possível que o sistema todo caia.

- **Acesso irrestrito**: usuários com acesso privilegiado (por exemplo, um administrador de rede insatisfeito com seu salário) podem fazer diversas "malvadezas" como reconfigurar serviços ou matar processos importantes, além de plantar cavalos de tróia, bombas de tempo e diversos tipos de vírus.

Há diversas ferramentas na Internet que se prestam a esse papel, e mesmo script-kiddies com um pouco mais de tutano podem criar ferramentas simples para tal. Quer um exemplo? Em uma máquina Windows, crie um arquivo DADOS.BAT e, dentro dele, coloque:

```
@ ECHO OFF
ECHO "Hacker" > DADOS.DAT
:VOLTA
TYPE DADOS.DAT >> SYSTEM.DAT
TYPE SYSTEM.DAT >> DADOS.DAT
GOTO VOLTA:
```

Rode o programa DADOS.BAT e você verá os arquivos DADOS.DAT e SYSTEM.DAT crescerem indefinidamente. Em um shell Unix é tão simples quanto. Experimente o seguinte script (chame-o de **dados.sh**):

```
#/bin/sh
touch dados.dat
touch system.dat
echo “hacker” >> dados.dat;
while [ 1 = 1 ];
do
cat dados.dat >> system.dat;
cat system.dat >> dados.dat;
done;
```

Execute o programinha (**dados.bat** no Windows, **sh dados.sh** no Unix) e veja o que acontece. Aparentemente nada, mas se você der um ls no Unix ou dir no DOS para listar os arquivos (em outro shell ou outra janela do DOS) verá que o tamanho de DADOS.DAT e SYSTEM.DAT crescem exponencialmente à medida que o tempo passa. Rodando por apenas um minuto (60 segundos exatos) os tamanhos ficarão da ordem de 100 MB cada. Mais dez minutos e teríamos dois arquivos de 1 GB cada, em 60 minutos (uma mísera hora) teríamos 12 GB ocupados. Os programas em DOS e Unix apresentaram desempenho semelhante quando executados no mesmo hardware.

Observe que nenhum dos 2 scripts precisou de privilégios especiais para rodar: foram executados diretamente na área autorizada para o usuário e usaram ferramentas e recursos disponíveis no sistema – nada precisou ser instalado. Observe ainda que os quatro problemas listados – apagamento ou destruição, consumo de recursos, vulnerabilidades locais e alterações por acesso irrestrito podem muito bem ser implementadas nos vírus que chegam a você pela Internet e por disquetes e CDs infectados!!! De fato, os vírus de computador, desde que surgiram no início dos anos 80, são um exemplo clássico de negação de serviço local!

Mas tudo isso é de muitíssimo baixo nível. Se algum sistema computacional está à mercê de operários braçais munidos de marretas e picaretas alguém deve perder o emprego – provavelmente o segurança da portaria **e** o CEO da empresa... Entretanto, há alguns tipos de ataques orquestrados externamente – em outra parte da rede interna da empresa ou mesmo via Internet.

Negação de serviço remoto

Além do anônimato inerente aos ataques remotos, há uma infinidade de métodos e ferramentas que permitem que um ataque remoto tipo DoS seja destruidor. Vamos nos ater mais à teoria sobre DoS e indicar algumas ferramentas para levar tais ataques a bom termo. Mas como é, de longe, a modalidade de ataque mais popular, documentada e comentada da Internet, deixaremos ao leitor a tarefa de buscar mais informações a respeito.

Há dois “subtipos” de ataques DoS remotos. Alguns deles atacam vulnerabilidades conhecidas nos sistemas-alvo – como as falhas de RPC que resultaram na queda de nove dos 13 roteadores-raiz da Internet (todos nos EUA) em outubro de 2002. O outro grande grupo de DoS remoto procura exaurir todos os recursos do alvo – seja ocupação de banda da rede, multiplicação descontrolada de processos no servidor HTTP ou inundação de mensagens de e-mail – possivelmente acompanhadas de vírus potencializadores do ataque, como o Klez e o BugBear. O famoso e ancião Ping of Death – tipo de ataque usando o utilitário Ping para gerar pacotes ICMP defeituosos e gigantescos – pertencia a ambos os tipos: gerava uma fragmentação defeituosa dos pacotes e ao mesmo tempo consumia banda.

Um exemplo de ferramenta de DoS remoto do primeiro tipo é o veterano WinNuke. O programa aproveitava uma falha existente no Windows 95 e no Windows NT 3.51: caso a porta 139 – nossa velha conhecida do compartilhamento de arquivos em redes Microsoft – recebesse pacotes não válidos em lugar do protocolo SMB, o sistema operacional travava. Há versões atuais do WinNuke que exploram outras vulnerabilidades – uma vez que essa foi consertada no Windows 98 e no NT4 – e trabalham também com exaustão de recursos.

Para saber mais sobre esse tipo de ferramentas, procure por **DoS** ou **malformed packet dos attack** em seu mecanismo de busca. Inúmeras ferramentas para esse e outros tipos de ataque podem ser encontradas em packetstormsecurity.nl/DoS/ e em www.astalavista.box.sk.

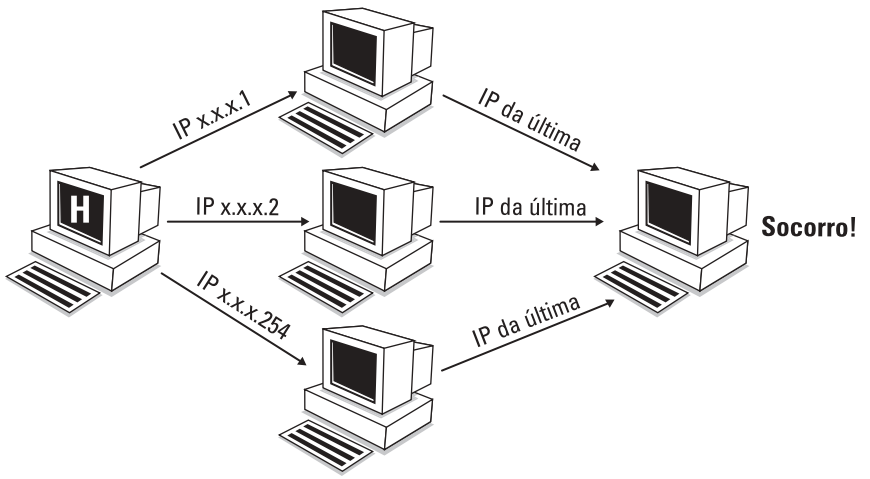
Já recaindo no segundo tipo, o tipo mais comum de ataque é o SYN Flood ou inundação de pacotes TCP SYN. Lembra-se de nossa primeira tentativa de spoofing? Enfiávamos um SYN com IP de origem diferente do nosso. Naquela situação, não conseguíamos estabelecer uma conexão spoofada por dois motivos:

1. Como a máquina atacada não sabia nosso IP verdadeiro, não poderia responder-nos;

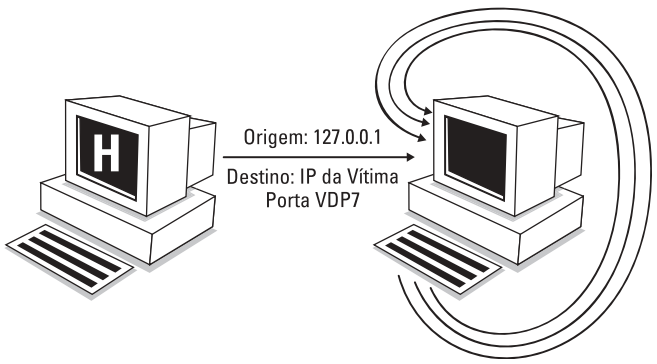
2. A máquina pela qual nos fazíamos passar retornava pacotes TCP RST e acabava com nossa conexão – e com nossa alegria...

No caso de um DoS por SYN Flood, de que nos importa se seremos respondidos ou não? O que vale é enviarmos tantos TCP SYN Spoofados quanto possível, para que a máquina-alvo se ocupe de respondê-los e não tenha tempo de responder a outras requisições de outras pessoas. Lembre-se: a vítima vai enviar um SYN-ACK e esperar por ACKs que nunca virão. Isso acabará por impossibilitar a vítima de responder a qualquer outro pedido de conexão ou mesmo atender a conexões já existentes.

Obviamente há outros métodos para DoS além do SYN Flood. Um outro método muitíssimo usado faz uso de pacotes ICMP Echo Request (ping!) disparados também em inundação. Se o IP estiver spoofado, a máquina-alvo não responderá ao IP de origem, mas ao IP falso. Uma extrapolação do DoS utilizando ICMP é o famoso Smurf: dispara-se ICMP Echo Request para um grande número de máquinas, mas colocando como endereço IP de origem o IP da vítima (e não um falso). O resultado é um grande número de máquinas refletindo ao mesmo tempo uma multidão de pacotes ICMP em direção à vítima, que sai do ar praticamente no mesmo instante.



Outra possibilidade é usar uma mensagem UDP em vez de ICMP. Há um serviço chamado echo, funcionando na porta 7 UDP, que simplesmente devolve ao endereço de origem tudo o que chegar por ela. Bem, imagine então enviar um pacote UDP spoofado, cujo IP de origem é 127.0.0.1 (ou seja, o loopback – em outras palavras, a própria máquina) e o IP de destino é o IP da máquina. Em uma situação dessas, o pacote UDP entra em loop dentro da máquina. Um número pequeno desses pacotes basta para comprometer toda a pilha TCP/IP do sistema operacional e consumir grandes percentagens da banda de rede disponível. Extrapolando da mesma forma como no ataque Smurf, pode-se enviar para a porta UDP 7 de inúmeras máquinas pacotes com IP de origem igual ao da vítima. Todas elas devolverão o presente para a máquina sob ataque – comprometendo a banda novamente. Se o *hacker* usar broadcast, então, o que era uma leve malvadeza passa a ser uma calamidade. Tal ataque é chamado de Fraggle.



Além desses métodos que tiram a máquina do ar, há outros tipos bem mais sutis, e mesmo alguns inusitados. Por exemplo, lembra-se de nossos programas de brute force? Alguns sistemas, para evitar justamente serem invadidos por brute force, limitam o número de logins permitidos (o número varia, mas a quantidade usual é três). Bem, um programa de brute force pode ser usado, então, para bloquear sistematicamente todas as contas do sistema, impedindo que qualquer usuário-se logue. Três acessos em cada conta são suficientes, o que torna a operação extremamente rápida e eficaz.

Pode-se também multiplicar conexões a um serviço específico até que todos os recursos do servidor em questão sejam consumidos – seja na forma de processos abertos, seja por meio de esgotamento de banda. Por exemplo, se quisermos derrubar um computador cujo servidor de Telnet esteja ligado, basta rodarmos um script como este em nossa máquina Unix:

```
while [1=1] ;
   do
      telnet ip.da.vítima.aqui &
   done;
```

É possível fazer o mesmo no Windows com um pouco mais de trabalho. Troque telnet pelo **lynx** e você tem uma ferramenta para causar o mesmo estrago em servidores HTTP. Troque pelo comando **mail** e você poderá encher o servidor SMTP, etc. Há ainda a possibilidade de fazer o mesmo com ping, Redes Microsoft, SSH, FTP (servidores FTP do Novell Netware são especialmente apetitosos...), Usenet, Finger, MS-SQL, MS-Access, ICQ... Acho que você já "pescou" a idéia.

Apesar dos ataques acima descritos poderem ser feitos manualmente por hackers de verdade, existem algumas ferramentas que facilitam e automatizam esse tipos de ataque – e os lammers as adoram. Tais ferramentas podem ser encontradas aos montes nos mesmos lugares anteriormente citados: packetstormsecurity.nl/DoS/ e em www.astalavista.box.sk. No endereço www.astalavista.com/library/ddos/basics/intro.shtml há um tutorial interessante (em inglês) sobre diversos tipos de pequenos ataques de DoS local e remoto que podem ser tentados. Estude todos eles, procurando na Internet exemplos e informações adicionais sobre cada um.

Mas há algo muito mais malvado que DoS simples: os ataques por negação de serviço remoto distribuído, ou DDoS.

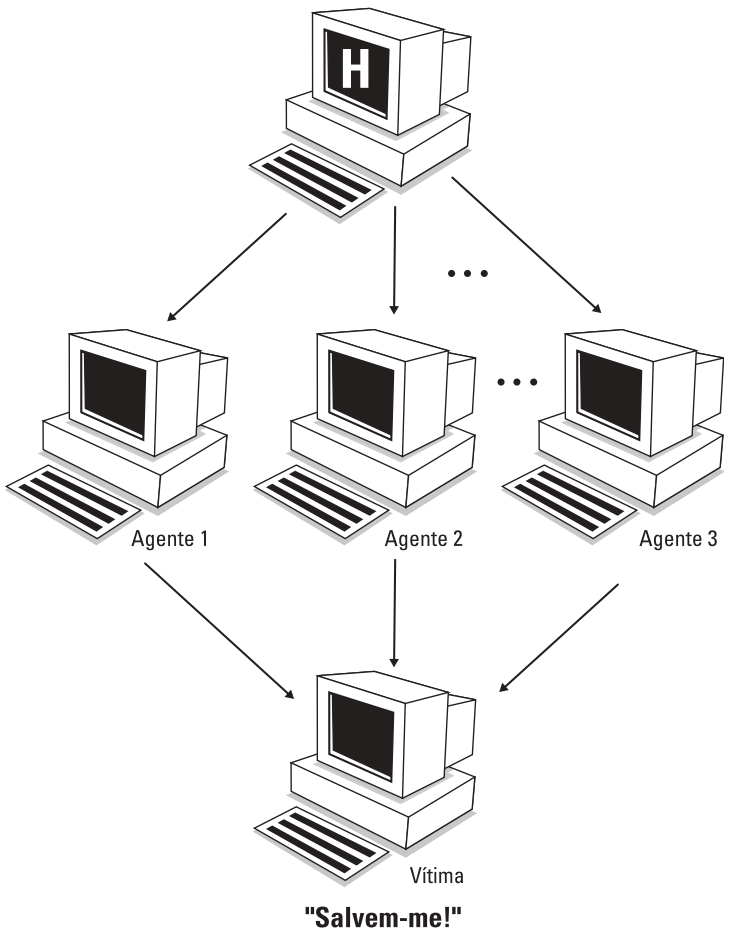
Negação de Serviço Remoto Distribuído

Os ataques Smurf e Fraggle já são maneiras efetivas de ampliar ataques originados em um único ponto – por isso mesmo são chamados de "lentes de aumento". Entretanto, mesmo se utilizando dessa lente, a banda de rede disponível para o atacante é limitada, e ele não pode manter um ataque com muitos

pacotes para cada máquina amplificadora e manter, ao mesmo tempo, muitas máquinas amplificadoras enviando esses mesmos pacotes à vítima.

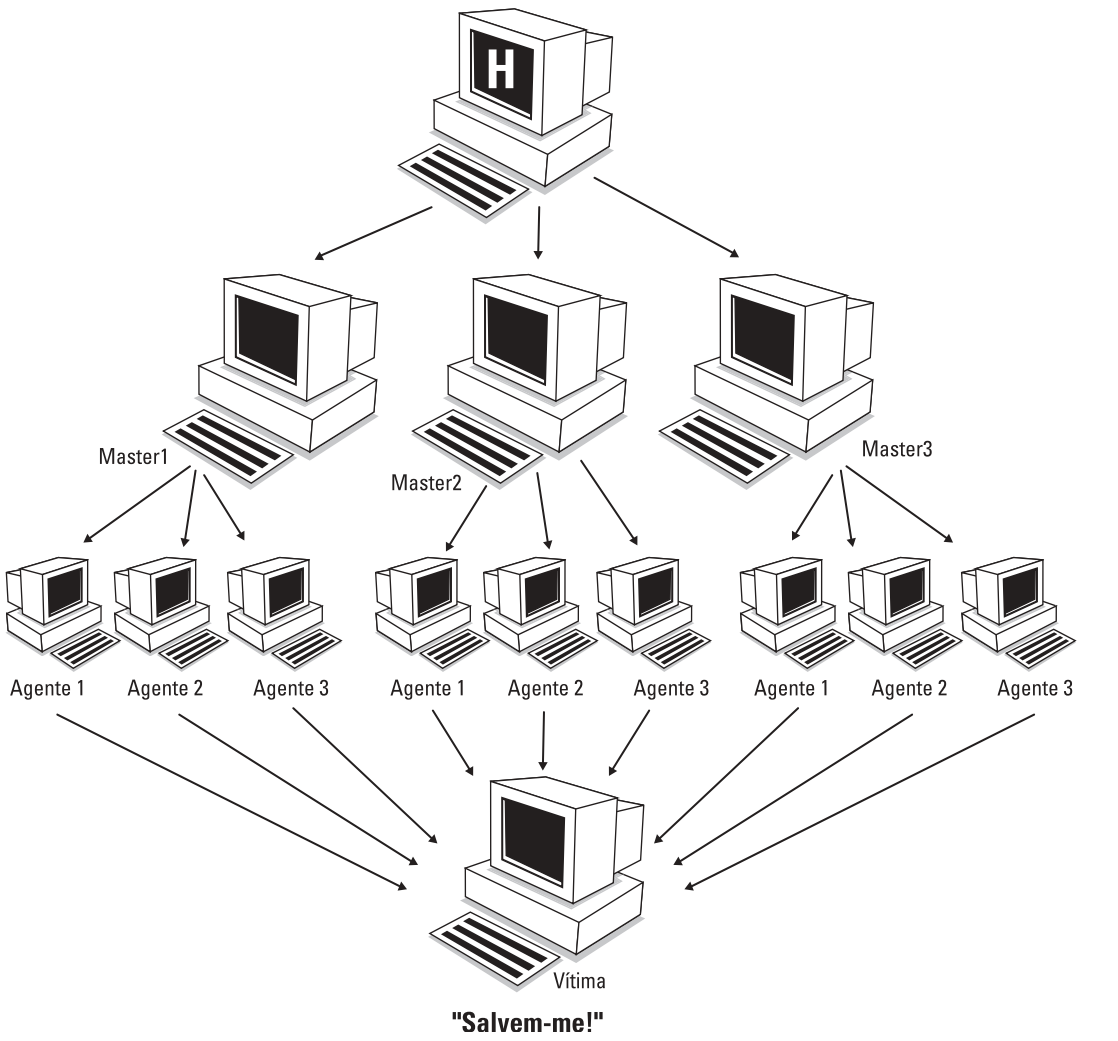
Para resolver a parada, foi desenvolvido um certo tipo de ataque em que os pacotes destinados às vítimas não saem da máquina do hacker, mas de computadores zumbis controlados remotamente por ele. Nesses computadores, o hacker instala certos tipos de cavalo de tróia ou vírus que respondem a comandos externos e funcionam como fonte geradora do ataque são os chamados **agentes** ou **zumbis**. O "contágio" dá-se pelos mesmíssimos métodos já estudados até aqui – invasão, distribuição camuflada e mesmo cooperação. O atacante possui um ou mais programas **mestre**, que controlam os agentes.

Lembra-se de quando dissemos que a invasão de sistemas menores poderia ser usado como trampolim para chegar à vítima principal? Pois bem, esta é uma das situações. O atacante leva semanas, às vezes meses, para invadir pequenos sistemas – computadores pessoais e servidores de pequenas empresas – sem que seu interesse imediato seja tais máquinas. Elas serão usadas, isso sim, como zumbis num ataque a grandes companhias (1234 1234 1234...), e para isso é necessário que os softwares adequados estejam instalados nelas. O hacker pode, inclusive, consertar algumas coisas e até atualizar a máquina do indivíduo, caso isso seja necessário para que o ataque principal seja levado a termo.

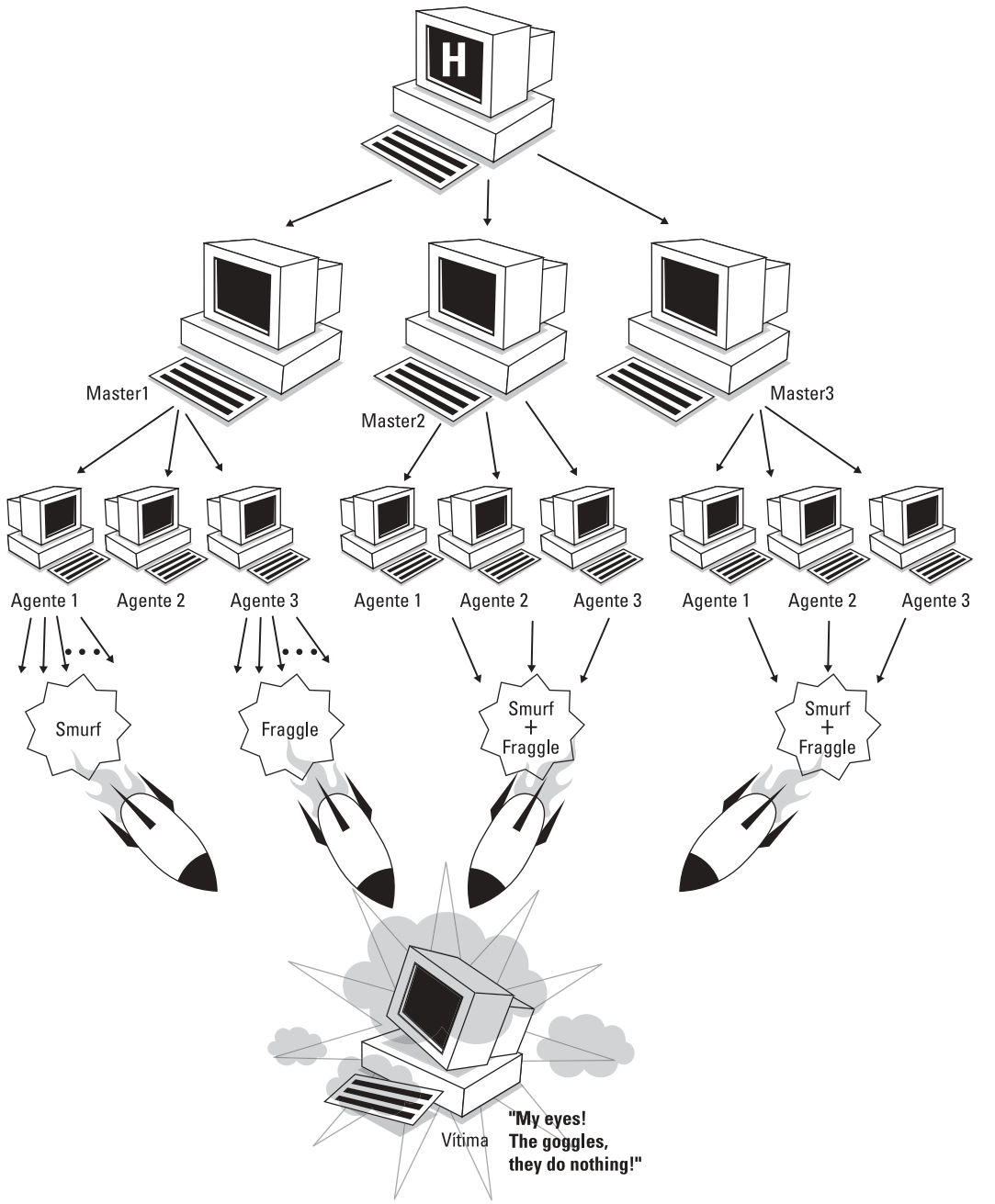


Uma vez instalado os agentes, o hacker pode iniciar um ataque a partir do mestre:

Imagine agora que o hacker é mais tarimbado. Um ataque DDoS pode ser sofisticado com a distribuição de diversos mestres, controlados também à distância pelo invasor. Ele pode tanto usar ferramentas especiais para tal, já existentes, como também usar coisas do sistema operacional como o Netcat ou o Telnet.



Agora, uma pitada a mais de maldade: cada um dos agentes pode usar lentes Smurf ou Fraggle para multiplicar sua força. Um ataque com essa configuração é devastador!



Masters and Slaves

Há uma infinidade de ferramentas para todos os ataques DoD e DDoS descritos aqui.A maioria pode ser encontrada nos dois sites indicados anteriormente: packetstormsecurity.nl/DoS/ e www.astalavista.box.sk. Outra seção, packetstormsecurity.nl/distributed/, trata de ferramentas e tutoriais DDoS.

Procure, nos sites indicados e em seu mecanismo de busca preferido, informações e download das seguintes ferramentas:

- Fapi
- Targa
- Blitznet
- Trin00/WinTrin00
- TFN/TFN2k/TFNWin
- Stacheldraht
- Shaft
- Trank
- Trinity

Como tarefa para casa, o estimado leitor terá que ler a documentação e testar todos, um a um, e descobrir em quais categorias (ás vezes mais de uma) cada um deles se encaixa.

Outra tarefa: procure informações sobre ataques a roteadores. Você ficará surpreso em saber que por vezes é mais fácil atacar equipamentos intermediários da Internet do que o alvo final. Se está muito difícil tirar do ar o servidor daquela empresa inimiga do meio-ambiente, talvez o roteador ligado a ele seja o lado mais fraco – e, de quebra, deixa a rede interna deles sem acessos à Grande Rede.

Defesa e contra-ataque

Bem, lá vamos nós. Este foi um capítulo longo. Procuramos colocar nele o básico do básico sobre ataques a computadores e redes. Listamos aqui, portanto, algumas dicas sobre o que procurar e por onde começar para segurar seus sistemas e evitar que sejam atacados com essas técnicas.

Recomendações aplicáveis a todos os tipos de ataques

O mais importante! Deixe ativados apenas os serviços que está realmente usando. Se o servidor é apenas de HTTP e FTP, desabilite Finger, Telnet, SSH, SMTP, POP, IMAP, Quake...

Configure corretamente seu firewall, e tenha sempre mais de um tipo, pelo menos um diferente para cada lado da DMZ. Prefira filtros de pacote por estado de conexão (Stateful Packet Filters) e firewalls do tipo Proxy. Mesmo com os serviços desabilitados nos servidores, feche as portas correspondentes no

firewall para evitar tráfego fantasma (por exemplo, de backdoors) e ACK Scanning. Não tenha preguiça: faça uma tabela de regras de filtragem realmente longa e abrangente, preferencialmente associando portas e endereços e não simplesmente bloqueando.

Mantenha sempre seus sistemas atualizados para evitar ser invadido por vulnerabilidades conhecidas e bem documentadas. Atenção especial aos hotfixes e Service Packs da Microsoft – o que não quer dizer que seus sistemas Unix e Novell precisam de menos cuidado.

Por padrão, crie usuários com o mínimo possível de privilégios e vá aumentando-os à medida que seja preciso. Caso o usuário não precise mais do privilégio, não hesite em cassá-lo. Crie políticas severas e consistentes de contas, que incluam nomes de login não-óbvios, gerenciamento ostensivo de contas ativas, desativação imediata (ou antes, cancelamento) de contas ociosas, senhas fortes contendo letras, números e símbolos e regras para alteração de senhas em menos de 30 dias. Além disso, um documento interno oficial, assinado pelo funcionário, deve regular claramente os equipamentos autorizados a conectar à rede e prever punições caso a rede seja invadida ou comprometida por mau uso ou descuido do usuário.

War Dialinge Brute Force

Em primeiro lugar, tente atacar a si mesmo com ferramentas de força bruta. Se você não deveria possuir modems ou acessos pessoais de alta velocidade ligados à sua rede, esta é uma boa maneira de descobrir. Ao menor sinal de modem discado, cable modem ou aDSL não autorizados em seu sistema, remova-os imediatamente!

Para sistemas acessíveis por Internet ou por conexão via terminal, a recomendação é não dar acesso a todas as contas por default. E cuidado com as contas padrão do sistema! Obrigatoriamente, os acessos deverão ser feitos por VPNs criptografadas.

No caso de seu negócio ou instituição realmente necessitar de modems e logins externos via Internet ou SSH, a política de senhas e acesso da empresa deve ser seguida à risca, e as penalidades aplicadas de forma exemplar.

Estouro de pilha

Não há muito o que dizer além do óbvio: se você é programador, sua obrigação é escrever código imune a buffer overflow. Mesmo sendo inerente às linguagens de programação, é possível implementar rotinas de verificação que barram injeção maliciosa de dados.

Se você é administrador de sistemas ou mesmo usuário doméstico, mantenha seu sistema sempre atualizado com os últimos "remendos" publicados pelos fabricantes de seu software. E não se concentre apenas nos servidores: as estações também são vulneráveis e portas de entrada para sua rede.

Quebra de senhas

Para começar, use ferramentas de quebra de senha em você mesmo. Só termine quando estiver satisfeito com o resultado – que deve ser, por acaso, zero: nenhuma senha a descoberto. Faça esse teste periodicamente, pelo menos com o dobro da freqüência com a qual as senhas devem ser mudadas. Se a política da empresa obriga os usuários a mudar a senha a cada 30 dias, tente quebrar as senhas do sistema pelo menos a cada 15 dias.

O método mais eficiente para barrar o brute force é limitando o número de logins e bloqueando **temporariamente** (e não em definitivo) as contas do sistema. Isso cria um problema de DoS, mas ainda é melhor que ter seus dados comprometidos.

Evite o roubo de senhas a todo custo, e dificulte ao máximo a vida do cracker que porventura conseguí-lo. Vale desabilitar os LM Hashes no Windows NT/2k, criar um servidor de login centralizado, com criptografia e segurança (como o Kerberos), usar Shadow Passwords no Unix (também com login centralizado via LDAP+Kerberos) e mesmo criptografar os sistemas de arquivos.

As severas políticas de senhas descritas acima também se aplicam neste caso. Bem como campanhas de consientização dos usuários para os males das senhas fracas e do comportamento de risco – senhas não devem ser escritas em lugar algum, muito menos divulgadas. NUNCA!

Por fim, a instalação de softwares que rejeitam senhas fracas no ato do cadastro do usuário (e posterior troca de senha periódica – sua empresa instituiu isso, não é?) é um ponto-chave para evitar que os usuários coloquem em risco toda a rede por colocar o nome do cachorro como senha pessoal.

War Driving

Criptografe sua rede e ative o WEP. Ponto final.

SQL Injections e Cookie Poisoning

Crie dispositivos de verificação em **todos** (repito: **todos**) os campos de **todos** (novamente repito: **todos**) os formulários do seu site na Internet. Bloqueie sumariamente caracteres perigosos como =, ‘ e “, *, % e _. Uma boa política é liberar **apenas** letras e números e bloquear todo o resto.

Use sempre o método POST para enviá-los ao script processador, uma vez que o método GET deixa informações importantes na URL. Se possível, criptografe os campos antes de enviá-los. Melhor ainda: use conexões HTTPS/SSL com certificados emitidos por empresas idôneas e criptografia forte.

No caso de cookies, faça um controle de sessão coerente, com session IDs preferencialmente criptografados e, se possível, dinâmicos – mudando a cada página acessada. Não se deve confiar apenas na criptografia do SSL: criptografe tudo várias vezes ANTES de enviar por HTTPS (que também é criptografado).

Sniffing, Spoofing e Hijacking

Em primeiro lugar, coloque filtros anti-spoof e detectores de sniffers em todos os pontos de entrada, saída e passagem (roteadores entre sub-redes) de sua rede. IDSs são bem vindos e grandes companheiros dessas ferramentas.

Mesmo não sendo impeditivo para a ação do hacker, é um agente complicador: instale switches e bridges em vez de hubs e segmente sua rede ao máximo. Além do benefício do desempenho, isso cria uma camada a mais de dificuldade para o invasor. Se possível, divida a rede em subredes e coloque roteadores com filtros de pacotes muito bem estruturados para interligá-las. Dependendo do número de pessoas, tempo, orçamento e tamanho da rede, é possível configurar **estaticamente** as tabelas MAC dos switches e bridges em cada uma de suas portas. Com isso, o equipamento fica imune a ARP Spoofing e MAC Flooding (mas não ao entupimento da rede ou mesmo à negação de serviço possivelmente provocados por eles). Mas prepare-se: é uma tarefa hercúlea...

Outro método para dificultar (e muito!) a ação dessas ferramentas é a criptografia sistemática de toda a rede. Toda ela. Use tudo o que estiver à mão, cada qual para sua função específica: PGP/GPG, IPSec, HTTPs, SSH (use **sempre** SSH versão 2!). Já vi projetos de VPNs com vários níveis de tunelamento, todos criptografados. Leve em conta o fato de que cada nível criptográfico tem um impacto negativo fenomenal no desempenho total da rede e use o bom senso para dosar performance e segurança..

Implemente DMZs não só entre a rede corporativa e a Internet (isso é miopia!), mas também diversos níveis de DMZs e mesmo DMZs interdepartamentais!

Verifique a predictabilidade dos números seqüenciais TCP de seus computadores. Máquinas Windows 9x tornam isso brinquedo de criança, máquinas FreeBSD e Solaris, por outro lado, são famosas por serem praticamente randômicas nesse ponto.

Se estiver usando sistemas Unix, esqueça os Unix Trusts. Além deles, outro esquema de confiança muito ruim é o que a Microsoft chama de **PDC/PDC Trusted Relations** no Windows NT. Evite-os completamente. Prefira sistemas modernos como o Novell eDirectory ou as inúmeras implementações do LDAP, incluindo aí o ADS da Microsoft. Mas use **sempre** um esquema de criptografia. Se nada disso estiver disponível, é mais seguro deixar o usuário logar-se individualmente em cada um dos sistemas da empresa do que ter um sistema de autenticação centralizado cujo esquema de confiança é falho.

Negação de Serviço

IDSs ajudam a detectar ataques DoS locais – a não ser que o ataque destrua o próprio IDS, o que é bem possível e mesmo provável.

As defesas contra IP Spoofing, ARP Spoofing e MAC Flooding também ajudam aqui. Tabelas MAC estáticas em switches idem.

Sistemas com a última atualização liberada pela fabricante são menos expostos a DoS baseados em vulnerabilidades conhecidas. Alguns sistemas possuem patches que os tornam inclusive imunes a ataques do tipo SYN Flood e Smurf.

Se possível, compre mais velocidade para sua conexão e tenha sempre rotas alternativas (e secretas!) caso a conexão principal esteja inundada. Adequação de tráfego (Traffic Shape) também é desejável.

Para o DDoS, uma única recomendação, além das anteriores: mantenha os zumbis longe de suas máquinas! Faça uma auditoria periódica em todos os computadores à procura de portas suspeitas ou programas não-autorizados.

Uma última dica: Echolot (echolot.sourceforge.net).