TÉCNICAS DE MONITORAÇÃO DE ATIVIDADES EM HONEYPOTS DE ALTA INTERATIVIDADE

Luiz Gustavo C. Barbato e Antonio Montes
Laboratório Associado de Computação e Matemática Aplicada
Instituto Nacional de Pesquisas Espaciais
CEP 12227-010 - São José dos Campos - SP
{lgbarbato,montes}@lac.inpe.br

RESUMO

Hoje em dia, o uso de honeypots vêm ajudando a melhorar a segurança de sistemas permitindo o aprendizado das técnicas adotadas pelos invasores com os próprios invasores. Com base neste novo recurso de segurança de sistemas de informação, este artigo apresentará técnicas de monitoração de atividades em honeypots de alta interatividade. Ferramentas baseadas nessas técnicas em user e kernel space também serão apresentadas.

ABSTRACT

Nowadays, the use of honeypots are helping to improve systems security, providing information about the techniques used by the attackers, from the attackers themselves. Based on this new resource for information systems security, this paper shows the techniques for activity monitoring in high-interaction honeypots. Tools based on these techniques in user and kernel space are also presented.

1 INTRODUÇÃO

Desde muito tempo atrás, estratégias militares [1] de defesa bem sucedidas são aquelas criadas com base no conhecimento do inimigo, seus tipos de armas, seus métodos de ataques, suas táticas de guerra e, principalmente, o seu objetivo. Conhecendo suas armas é possível conduzir treinamentos práticos. Conhecendo seus métodos de ataque é possível desenvolver métodos de defesa. Conhecendo suas táticas de guerra é possível projetar mecanismos para combatê-lo. E conhecendo seus objetivos é possível entender a motivação que o incentivou a iniciar os ataques.

Assim como os militares, a área de segurança de sistemas de informação também pode utilizar estes conhecimentos análogos para se defender contra ataques. Esta busca motivou os profissionais de segurança a criarem metodologias para obter informações mais precisas sobre os atacantes[1].

Uma das metodologias criadas faz uso de ferramentas de pesquisa, conhecidas como *honeynets*, que permitem a captura e estudo de cada passo dos atacantes. *Honeynets* são redes preparadas para serem comprometidas, compostas por várias máquinas e equipamentos denominados *honeypots*. Em cada *honeypot* são instalados sistemas e aplicativos reais, com intuito de criar ambientes similares aos que são utilizados em muitas organizações atualmente. Além dos sistemas e aplicativos, outros recursos são necessários para monitorar as operações efetuadas pelos atacantes durante um processo de invasão.

2 HISTÓRICO

2.1 Honeypots

A primeira ocorrência publicada sobre monitoração dos passos dos atacantes ocorreu em 1988, quan-do Clifford Stoll [2] relatou um ataque sofrido pelo Lawrence Berkeley Laboratory(LBL). Um ano mais tarde, este relato se tornou um livro [3]. Nesta invasão sofrida pelo LBL, ao invés de tentarem bloquear os intrusos, Stoll e sua equipe resolveram permitir o acesso ao sistema enquanto monitoravam todas as atividades dos atacantes até que suas origens fossem descobertas, o que levou quase um ano. Para acompanhar os passos dos atacantes, impressoras foram instaladas em todas as portas de entrada do sistema, configuradas para imprimir todas as atividades que os atacantes realizavam nas máquinas. Com a ajuda dessas informações, foi possível identificar não só a origem dos invasores, mas também quais eram seus objetivos e motivações, quais redes foram comprometidas e quais redes os invasores estavam interessados em atacar.

Em 1992, Bill Cheswick [4] publicou um artigo descrevendo o acompanhamento de uma invasão ocorrida em 7 de janeiro de 1991 no Laboratório Bell da AT&T. Esta invasão ocorreu no *gateway* do laboratório que estava preparado para ser comprometido, rodando várias aplicações falsas tais como: FTP, telnet, SMTP, finger e rlogin. O intuito desta monitoração era descobrir quem estava interessado em atacar o laboratório; de onde vinham e qual a frequência destes ataques; e em que tipos de vulnerabilidades os atacantes estariam mais interessados. Esta monitoração durou meses, e contou com a ajuda de várias pessoas como Steven M. Bellovin [5] que desenvolveu várias ferramentas utilizadas para atrair, conter e capturar as ações dos invasores.

Em 2002, Lance Spitzner [6] definiu claramente o conceito de *honeypot* como sendo "*Um recurso de segurança preparado para ser sondado, atacado ou comprometido*", e explicou a sua real função "*Honeypots não são uma solução, eles não consertam nada, são apenas uma ferramenta. E o seu uso depende do que se deseja alcançar*".

Em 2003, Martin Roesch [7], o criador do Snort, categorizou o uso dos *honeypots* em dois tipos: produção ou pesquisa. Para Roesch, *honeypots de produção são utilizados para diminuir os riscos e ajudar a proteger as redes das organizações; e os honeypots*

[1]Neste artigo, os termos atacantes e invasores possuirão o mesmo significado. Serão interpretados como sendo indivíduos mal intencionados que atacam e/ou invadem sistemas.

de pesquisa são designados para estudar e obter informações da comunidade dos atacantes.

Os *honeypots* também podem ser classificados como de baixa(serviços falsos) ou alta interatividade (serviços reais), nos quais os atacantes podem obter acesso total ao sistema. Os *honeypots* de alta interatividade, foco do artigo, não emulam serviços, ao invés disso, são instalados sistemas operacionais e aplicativos reais para os invasores interagirem com a máquina e permitir que seus passos sejam acompanhados.

2.2 Honeynets

A idéia de se criar uma rede preparada para ser comprometida visando o aprendizado, começou com uma pequena iniciativa de um analista de segurança da Sun Microsystems, chamado Lance Spitzner [8], que resolveu conectar uma máquina Linux Red Hat 5.0 à Internet em 25 de fevereiro de 1999. Sua intenção era fazer com que a comunidade dos atacantes lhe mostrasse como agia para rastrear, atacar e explorar sistemas.

Surpreendentemente, sua armadilha funcionou em 15 minutos, quando sua máquina Linux foi invadida com sucesso. Infelizmente, o atacante percebeu que estava sendo monitorado e apagou todo o conteúdo do sistema, destruindo todos os *logs* e ferramentas utilizadas durante o ataque.

Em consequência disso, Spitzner resolveu preparar todo um ambiente para capturar os dados, de for-ma que os atacantes não conseguissem mais apagá-los.

Com a evolução desta idéia surgiu em 25 de abril de 1999 o *Honeynet Project*[2], que inicialmente foi formado por 30 profissionais interessados em aprender sobre as ferramentas, táticas e motivações dos atacantes, e além disso compartilhar estas informações visando o aumento da segurança de sistemas.

Só em junho de 2000 o projeto entrou em operação efetiva, quando um dos *honeypots* foi invadido. A invasão ocorreu em um Solaris 2.6 por um grupo organizado de atacantes, tendo sido necessário contar com as habilidades de todos os integrantes do projeto para capturar os dados do ataque.

Esta experiência motivou o grupo a desenvolver uma ferramenta denominada de *honeynet*, que nada mais é que uma rede preparada para ser comprometida, ou seja, uma espécie de laboratório que fica exposto à Internet para ser atacado, porém com alguns mecanismos de monitoração, coleta e contenção de dados.

Com o passar do tempo, foram definidos dois elementos cruciais para a criação e manutenção bem sucedida de uma *honeynet* [8]. São eles: contenção e captura de dados. A contenção de dados foi definida como sendo uma forma de impedir que os atacantes, após invadirem a *honeynet*, a utilizem para atacar outras redes e a captura de dados foi definida com sendo uma maneira de coletar o maior número possível de informações, com o objetivo de promover um estudo detalhado dos passos dos atacantes.

[2]http://www.honeynet.org

Com a finalidade de juntar várias instituições internacionais envolvidas com pesquisas de *honeynets* foi criada a *Honeynet Research Alliance*[3]. Um dos projetos que faz parte do *Honeynet Research Alliance* é o Honeynet.BR[4] [9], criado em dezembro de 2001 no INPE com intuito de acompanhar as atividades maliciosas na Internet brasileira.

3 MONITORAÇÃO DE ATIVIDADES

Como citado anteriormente, as *honeynets* são uma ferramenta composta por várias máquinas e equipamentos denominados *honeypots*, e outros recursos para a coleta e contenção dos dados como: *firewalls*, sistemas de detecção de intrusão, *loghosts*, roteadores, entre outros.

Para estudar atividades hostis em *honeypots* de alta interatividade é preciso que se instale um mecanismo de coleta de dados em todas as máquinas sob a forma de sistema de monitoração. Este mecanismo de monitoração deve permanecer escondido, de forma a não deixar que o invasor descubra que está sendo vigiado, colocando em risco toda a integridade do projeto de pesquisa.

Nas primeiras implementações de *honeynets*, este tipo de monitoração era feito diretamente pelos sistemas de detecção de intrusão, visto que este está implantado na rede em pontos estratégicos para capturar e analisar todo o tráfego. Com essa estrutura, é possível remontar, por exemplo, todos os comandos e respostas em uma transação FTP, ou uma sessão telnet, ou em uma conversa de IRC. O sistema de detecção de intrusão oferece esta facilidade porque essas informações trafegam em claro pela rede, ou seja, informações não criptografadas.

No entanto, esta topologia apresenta problemas no caso do uso de conexões criptografadas, como um SSH por exemplo. Com isso nem todas atividades dos atacantes podem ser monitoradas coletando-se o tráfego de rede. Isso tem se tornado cada vez mais comum e para monitorar todas atividades dos invasores é necessário utilizar ferramentas que permitam a captura das ações dos atacantes antes que estas sejam criptografadas.

Basicamente existem duas formas de capturar estas operações:

1. captura em *user space*[5]
2. captura em *kernel space*[6] [10]

A captura em *user space* é baseada em versões modificadas de comandos importantes do sistema, como os interpretadores de comandos: *bash, ksh, csh, tcsh, sh*, etc.

[3]http://www.honeynet.org/alliance

[4]http://www.honeynet.org.br

[5]Espaço de memória destinado a execução de aplicativos dos usuários.

[6]Espaço de memória reservado a execução de códigos em modo privilegiado do processador, com acesso total a todos os recursos físicos de computador.

E a captura em *kernel space* pode ser feita utilizando módulos de *kernel*(Loadable Kernel Modules ou *LKMs*) [11] para alterar a funcionalidade do sistema, ou versões modificadas do próprio kernel [12].

3.1 Captura em User Space

A captura de atividades em *user space* pode ser feita através da modificação dos interpretadores de comandos, bibliotecas de sistema como a *libc*, ou com a ajuda de utilitários como o comando *script*.

3.1.1 Interpretadores de comandos

A idéia inicial de monitorar as atividades nos *honeypots* utilizando uma versão modificada do interpretador de comandos *bash* foi proposta por Antonomasia[7]. Nesta solução os dados são enviados para a aplicação *Syslog*. Tempo depois Anton Chuvakin[7] acrescentou a funcionalidade de retirar os dados dos *honeypots* através de pacotes UDP, não cifrados, ao invés de depender do *Syslog*.

Os interpretadores de comandos podem ser utilizados em monitoração, visto que eles são necessários nos sistemas operacionais para que os comandos sejam executados. Resumidamente, um interpretador de comandos é um processo pai que se divide ao ser invocado na execução de um comando/aplicação/*script* através da entrada padrão ou de um arquivo. Ao receber um pedido de execução, este se divide através da chamada de sistema *fork*, criando um novo processo referente ao comando/aplicação/*script* executado.

Com base nesta característica de execução dos interpretadores de comandos, estes podem ser modificados para antes de criarem um novo processo, registrar o que foi pedido para ser executado.

Uma preocupação relevante neste tipo de monitoração é que o invasor pode substituir o interpretador de comandos burlando todo o processo de monitoração.

Como exemplo de uma técnica de monitoração em *user space* utilizando os interpretadores de comandos, mostraremos na próxima seção as modificações que devem ser feitas no interpretador de comandos *bash*, para que ele possa ser utilizado na monitoração de atividades em *honeypots*.

3.1.2 Utilitário Script

O utilitário *script* é um aplicativo, encontrado na maioria dos UNIX, utilizado para registrar em um arquivo tudo que é impresso em um terminal. Por *default* todas as informações são gravadas em um arquivo local, o que não é interessante em *honeypots* [8]. Uma solução para este problema foi apresentada por Ryan C. Barnett[8] que inseriu a chamada do aplicativo *Cryptcat* no código fonte do *script* para transmitir estes dados, de forma cifrada, para uma outra aplicação remota.

[7] http://www.honeynet.org/papers/honeynet/tools

[8] http://honeypots.sourceforge.net/modified_script.html

Uma outra técnica para utilizar o comando *script* como mecanismo de monitoração necessita de uma pequena modificação no código fonte do mesmo. Esta modificação deve ser feita no arquivo `misc-utils/script.c` do pacote *util-linux* em máquinas linux.

```
void
dooutput() {
  ...
  for (;;) {
    ...
    cc = read(master, obuf, sizeof (obuf));
    ...
    (void) write(1, obuf, cc);
    //(void) fwrite(obuf, 1, cc, fscript);
    oknSendCmd ( obuf, cc );
    ...
  }
  ...
}
```

A função *oknSendCmd* é responsável por retirar as informações dos *honeypots*, de forma cifrada, e enviá-las para uma estação de monitoração. Esta função será descrita com mais detalhes posteriormente.

Para que este processo de monitoração seja iniciado, o aplicativo *script* precisa ser executado. Uma forma de fazer isso é invocá-lo nos arquivos de inicialização do *bash*, de maneira a executar o comando *script* toda vez que o interpretador de comandos for chamado.

3.1.3 Bibliotecas de Sistema

Uma outra forma de registrar as atividades dos invasores em *user space* é modificar a função resposável por executar os comandos na biblioteca *glibc*. Essa biblioteca é a interface de comunicação entre *user space* e *kernel space*, a responsável por gerar interrupções de *software* antes dessa transição iniciar.

A modificação feita no arquivo `sysdeps/unix/sysv/linux/execve.c` no pacote *glibc* envia para um estação de monitoração remota tudo que for pedido para ser executado no *honeypot*.

```
int
__execve (file, argv, envp)
     const char *file;
     char *const argv[];
     char *const envp[];
{ ...
  for (v = argv; *v; v++)
  // ;
    strncat ( cmd, *v, LOGSIZE - \
    strlen ( cmd ) );

  oknSendCmd ( cmd, strlen ( cmd ) );
  ...
}
weak_alias (__execve, execve)
```

Esta monitoração só funcionará se os aplicativos utilizarem dinamicamente essa biblioteca(uso dinâmico da biblioteca) ou se os aplicativos forem compila-

dos e linkados com esta após a modificação da mesma (uso estático da biblioteca).

3.1.4 Discussão

Uma outra preocupação na monitoração em *user space* é com relação a utilização de arquivos executáveis que dependem de bibliotecas dinâmicas, para registrar as atividades dos atacantes. Estas podem ser substituídas, alterando o comportamento destes binários.

Neste caso é interessante que as ferramentas utilizadas na monitoração sejam compiladas estaticamente. Desta forma todas as bibliotecas necessárias são anexadas junto ao executável. Compilando estaticamente também facilitaria a instalação destes mecanismos de monitoração nos *honeypots* que rodam o mesmo sistema operacional, pois neste caso é necessário apenas copiá-los para o local apropriado.

Logo após a compilação estática das ferramentas é importante a eliminação de todos os símbolos desnecessários, com a ajuda do comando *strip*. Com isso, além de retirar certas informações que podem ajudar a descobrir o que a ferramenta faz, através do comando *strings*, o binário fica com um tamanho bem reduzido.

A principal vantagem da captura em *user space* é a simplicidade e rapidez de ser aplicada, além de poder ser utilizada em vários sistemas operacionais diferentes.

Em contrapartida, o *user space* é, em sua totalidade, dependente do *kernel space*, pois manipula informações recebidas do *kernel*. Sabendo disso, um atacante pode falsificar informações no *kernel space* e consequentemente enganar um mecanismo de monitoração em *user space* [13].

Para que o *user space* possa interagir com o *kernel space*, é necessário a utilização de um recurso conhecido como chamada de sistema(*system call*) [14]. As chamadas de sistema são responsáveis por passar informações, como funções e parâmetros de funções, a uma linguagem de mais baixo nível, reconhecida pelo *kernel*. Alterando essas chamadas de sistemas todas as informações em *user space* podem ser escondidas e/ou modificadas, alterando assim o comportamento dos mecanismos de monitoração.

3.2 Captura em Kernel Space

Essas dificuldades de monitoração em *user space* podem ser superadas com a utilização de técnicas que atuem diretamente no *kernel*.

No Linux, a captura [15] destas atividades pode ser feita através de:

- Módulos de kernel[9] interceptando[10] a função responsável por processar os *scancodes* (*handle_scancode*).
- Módulos de kernel interceptando a função responsável por inserir os caracteres na fila de *tty* (*put_queue*).
- Módulos de kernel interceptando a função responsável por receber os caracteres da fila de *tty* e colocá-los na fila de leitura(*receive_buf*).
- Módulos de kernel interceptando a função responsável por disponibilizar os caracteres para os processos(*tty_read*).
- Módulos de kernel interceptando a chamada de sistema responsável pela execução de comandos(*sys_execve*).
- Módulos de kernel interceptando as chamadas de sistema de leitura(*sys_read*) e escrita (*sys_write*).

[9]Todas essas implementações com módulos de kernel, podem ser feitas diretamente no código fonte do mesmo.

[10]Neste contexto, interceptar significa encontrar o endereço e modificar o comportamento da função.

3.2.1 sys_read e sys_write

A monitoração mais comum utilizando módulos é feita através da modificação do comportamento das chamadas de sistema *sys_read* e *sys_write*. A chamada de sistema *sys_read* é responsável pela leitura de todos os *file descriptors* do Linux, incluindo leitura de arquivos, leitura de *sockets* e leitura de teclas pressionadas. Com base nesta última característica, esta chamada de sistema é alterada para registrar todas as teclas pressionadas pelos atacantes. Já a *sys_write* é responsável pela escrita dos dados nos *file descriptors*, como escrita em arquivos, escrita em *sockets* e escrita de dados no terminal. Com a alteração dessas duas chamadas de sistema é possível remontar as atividades dos atacantes.

Resumidamente este algoritmo de captura seria:

```
int oknRead ( unsigned int fd, char * buf, \
size_t count )
{ ...
  if ( fd == 0 )
    oknLog ( buf, count, 0 );
  ...
}
int oknWrite ( unsigned int fd, char * buf, \
unsigned int count )
{ ...
  if ( fd == 1 )
    oknLog ( buf, count, 1 );
  ...
}
int init_module ( void )
{ ...
  orgRead = sys_call_table [ SYS_read ];
  sys_call_table [ SYS_read ] = oknRead;

  orgWrite = sys_call_table [ SYS_write ];
  sys_call_table [ SYS_write ] = oknWrite;
  ...
}
```

Esta técnica foi utilizada na ferramenta *sebek* desenvolvida por Edward Balas[7] e vem sendo utilizada pelo *Honeynet Project* para capturar as teclas pressionadas pelos invasores.

3.2.2 *tty_write*

Uma outra técnica que está sendo estudada com mais detalhes é a utilização da função *tty_write* na captura dessas informações. Essa técnica está sendo aplicada em um sistema que está sendo desenvolvido com intuito reconstruir todas as atividades de um invasor em uma máquina comprometida.

```
ssize_t oknTtyWrite ( struct file * filp, \
char * buf, size_t count, loff_t * ppos )
{ ...
  if ( buf )
    oknLog ( buf, ret );
  ...
}
int oknInitTty ( void )
{ ...
  filp = filp_open ( "/dev/tty0", \
  O_RDONLY, 0 );
  ...
  orgTtyWrite = filp->f_op->write;
  filp->f_op->write = oknTtyWrite;
  ...
}
int init_module ( void )
{ ...
  oknInitTty ( );
  ...
}
```

Neste caso estamos alterando a operação [11] de escrita que pode ser feita em um terminal. Essas operações são definidas em `<linux/fs.h>` e são acessadas tanto em um terminal, arquivo ou dispositivo. Como a operação *write* é utilizada para enviar dados para um dispositivo, substituimos esta função por uma outra que além de exercer a operação de escrita, nos permite registrar todos os dados que passam pelo terminal. Apesar de ter sido usado o dispositivo *tty0*, esta técnica permite a monitoração de qualquer terminal.

Essa técnica permite a monitoração em *kernel space* da mesma forma que faz o aplicativo *script*, pois tanto a entrada quanto a saída de dados no terminal é registrada. A principal vantagem dessa dessa técnica é fato dos dados já estarem transformados em formato *ASCII*.

3.2.3 *sys_execve*

Esta chamada de sistema é a responsável por requisitar ao *kernel* a execução de algum comando. Esta é invocada pela biblioteca de sistema *libc* através da função *execve*(interrupção de *software*), a mesma descrita na monitoração em *user space*.

Interceptando esta chamada de sistema é possível capturar as mesmas informações que as técnicas em *user space*(interpretadores de comandos e bibliotecas de sistema), porém com as vantagens de programação de *kernel* mencionadas abaixo.

```
int oknExecve ( struct pt_regs regs )
{  ...
   cmd = getname ( ( char * ) regs.ebx );
   ...
   oknLog ( cmd );
   ...
}
int init_module ( void )
{ ...
  orgExecve = sys_call_table [ SYS_execve ];
  sys_call_table [ SYS_execve ] = oknExecve;
  ...
}
```

A desvantagem desta técnica é que todos os comandos executados no sistema, tanto pelos usuários quanto pelo próprio sistema operacional será registrado, causando assim um volume muito grande de informações.

3.2.4 *Discussão*

Monitoração de atividades em *kernel space* tem como principal vantagem a capacidade de registrar todas as teclas pressionadas bem como o retorno dos comandos executados, de uma forma praticamente invisível para os invasores, pois todos os processos, arquivos utilizados, etc, são escondidos pelo próprio módulo/*kernel*. O *kernel* também se encarrega de não permitir que a monitoração seja interrompida.

Em contrapartida, não é uma solução simples de ser implementada e portada para outros sistemas operacionais devido as particularidades de cada arquitetura. Cada sistema operacional deverá possuir seu próprio mecanismo de coleta de dados.

4 MODIFICAÇÃO NO BASH

Esta solução foi desenvolvida com intuito de monitorar de forma simples e rápida os comandos executados pelos invasores em máquinas linux, freebsd e openbsd. Neste caso o interpretador de comandos *bash* foi escolhido pois pode ser encontrado em todos esses sistemas operacionais. Todos os códigos/ferramentas que serão mostrados abaixo podem ser utilizados nesses três sistemas.

Aqui é apresentada uma modificação no *bash*, mas a mesma pode ser feita em outros interpretadores de comandos como *sh*, *csh*, *ksh*, etc, variando apenas o local onde a função de captura, *oknSendCmd*, será inserida.

A única alteração feita no código fonte do *bash* foi a inserção da chamada da função *oknSendCmd*, na função *bash_add_history* que se encontra no arquivo *bashhist.c*.

```
void bash_add_history ( char * line )
{ ...
  oknSendCmd ( line, strlen ( line ) );
  ...
}
```

A inserção foi feita nesta função, porque esta possui a finalidade de registrar os comandos executados no arquivo de histórico, normalmente *bash_history*.

Então, antes de gravar os comandos no arquivo de histórico a função *oknSendCmd* os enviará para a máquina de monitoração.

A função *oknSendCmd* após receber os dados da *bash_add_history*, captura o horário em que o comando foi executado, o número de identificação do processo(interpretador de comandos) e o número de identificação do usuário que executou os comandos, para depois repassá-los para uma outra função, *oknCrypt*, que cifrará todos estes dados antes de serem enviados pelos *honeypots*. Os dados serão transmitidos através de pacotes UDP [16], devido as características simples de transporte de dados deste protocolo.

```
int oknSendCmd ( char * cmd, int size )
{ ...
  snprintf ( log, LOGSIZE, "%sTS=%s UID=%d \
  PID=%d CMD=%s\0", TAG, ts, getuid ( ), \
  getpid ( ), cmd );

  oknCrypt ( log+TAGLEN, 1, strlen ( log ) );

  if ( sendto ( ..., log, ... ) ... )
  ...
}
```

O algoritmo de criptografia utilizado é bem simples :

```
int oknCrypt ( char * message, int crypt, \
int size )
{ ...
  unsigned long func_sem = ( ( ( sem [ 0 ] \
  + sem [ 1 ] ) * sem [ 2 ] ) ^ sem [ 3 ] );
  ...
  while ( pos < size )
  {
    if ( crypt )
      message [ pos++ ] += ( sem [ pos % 4 ] \
      + func_sem + pos );
    else
      message [ pos++ ] -= ( sem [ pos % 4 ] \
      + func_sem + pos );
  }
  ...
}
```

O intuito da criptografia é apenas iludir os invasores, não deixando as informações trafegarem em claro pela rede. Para isso foi aplicado o método de substituição simples, que é rápido e atende as necessidades do momento.

Basicamente esse algoritmo é baseado na permutação fixa de *func_sem* que é somando a cada posição da mensagem junto com outros valores(*sem [pos % 4] + pos*). E para decifrar basta apenas diminuir esses valores de cada posição da mensagem. A chave de criptografia desta função é *sem*, um inteiro de 4 *bytes*.

Se passado um valor diferente de 0(zero) como segundo argumento desta função, esta cifrará os dados(somará), caso contrário os dados serão decifrados(diminuirá). Esta mesma função é utilizada na ferramenta de captura de dados que será descrita.

4.1 Ferramentas Auxiliares

4.1.1 Alteração do patch

Para alterar o valor das variáveis do *patch* criado para o *bash* foi desenvolvido uma ferramenta chamada *oknalterpatch*. Esta ferramenta recebe como argumentos o número IP(-i) da máquina de monitoração, o número da porta(-p) referente ao serviço, e o *patch*(-b) a ser alterado.

Exemplo:

```
$ oknalterpatch -i 10.0.0.13 -p 345 \
-b bash-2.05b-okn-patch
```

Essa ferramenta irá alterar o endereço IP e o número da porta na função *oknSendCmd*.

```
int oknSendCmd ( char * cmd, int size )
{ ...
  host.sin_addr.s_addr = 218103818;
  host.sin_port = 22785;
  ...
}
```

Os valores armazenados no *patch* são do tipo inteiro para ocultar o endereço de rede da máquina de monitoração, que poderia ser obtido através da utilização do comando *strings*.

4.1.2 Captura dos dados

A captura dos dados enviados pelos *honeypots* não é feita por uma aplicação servidora UDP, pois este tipo de solução implicaria em mais um ponto vulnerável não desejável na *honeynet*. Além do mais, seria necessário que a estação de monitoração possuísse um endereço de rede.

A estação de monitoração atua na camada 2 em relação do modelo OSI, ou seja, a estação está configurada para trabalhar em modo promíscuo. E para que os dados exportados pelos *honeypots* possam ser capturados, a aplicação de captura de dados, *oknbashsniff*, atua como monitorador de tráfego de rede, analisando tudo que passa no barramento.

A aplicação de captura de dados, *oknbashsniff*, identifica os dados exportados pelos *honeypots* baseado em quatro fatores:

1. Endereço IP de origem dos pacotes
2. Endereço IP de destino dos pacotes
3. Número de porta de destino dos pacotes
4. Uma *TAG* no início da área de dados dos pacotes

Os três primeiros itens podem ser filtrados através de uma expressão em BPF [17], do mesmo tipo que as utilizadas em outros monitores de tráfego de rede como o *tcpdump*, *ngrep*, e *ethereal* pois a monitoração é feita com a biblioteca *libpcap*. Este filtro pode ser passado para o *oknbashsniff* atráves da opção(-b).

O último item pode ser identificado através da opção(-t) do *oknbashsniff*. O uso desta *TAG* elimina muito os falso-positivos, ou seja pacotes UDP para a mesma máquina de destino, para o mesmo número de porta, cujo endereço de origem seja o mesmo dos *honeypots* e que não sejam referentes aos dados exportados pelo *bash*. Assim esta ferramenta consegue identificar quais os pacotes que são realmente referentes ao *bash*. Lembrando que esta *TAG* não elimina por completo os falsos-positivos sendo que pacotes UDP falsos podem ser criados com esta mesma *TAG* no início da área de dados.

É importante que esta *TAG* seja a mesma compilada no *bash*. Este valor pode ser facilmente alterado no *patch* para o *bash* através da opção(-t) da ferramenta *oknalterpatch*.

A ferramenta *oknbashsniff* também possui outras funcionalidades interessantes tais como:

- Filtros de conteúdo através de expressões regulares
- Entrada de dados através de arquivos no formato *tcpdump*
- Saída de dados em arquivos no formato *tcpdump*

Com os filtros de expressão regular(-r) é possível procurar por padrões na área de dados dos pacotes gerados pelo *bash*. Dentre estes padrões pode-se incluir, data e hora dos comandos executados, determinados comandos executados, determinadas ferramentas utilizadas, etc.

Com a entrada de dados de arquivos gerados no formato *tcpdump*(-i) é possível utilizar outros aplicativos para monitorar o tráfego de rede, como o próprio. O *tcpdump* também possui a funcionalidade de gravar os dados em arquivos, logo o *oknbashsniff* po-de ser utilizado apenas para analisar os dados gravados por aquele.

Da mesma forma que *oknbashsniff* lê arquivos do formato *tcpdump*, ele também pode gravar arquivos neste formato(-o), filtrando e registrando somente os pacotes exportados pelo *bash*.

4.1.3 Simulação de serviço

Como a aplicação que recebe os dados atua como monitorador de tráfego de rede, é necessário que alguma outra aplicação simule a existência do serviço referente ao número da porta configurada no *bash*, e que possua o mesmo endereço de rede(IP). Esta preocupação se deve ao fato que de acordo com o protocolo UDP [16], quando não há um serviço requisitado na máquina de destino, esta retorna um pacote ICMP do tipo *port unreachable*. E se o endereço de destino dos pacotes do *bash* não for encontrado na rede, o *honeypot* ficará emitindo pacotes ARP no barramento, em busca do endereço físico da máquina de destino.

Para sanar este problema, uma outra ferramenta simples foi desenvolvida, *oknsimserv*, para simular este serviço.

```
$ oknsimserv <número-porta>
```

Esta ferramenta é uma aplicação servidora UDP [18] que rodará automaticamente em *background* ignorando tudo que for recebido, por que neste caso o conteúdo dos pacotes é irrelevante. É interessante que esta ferramenta rode na máquina que atuará como *loghost*, porque este *honeypot* é considerado o mais seguro da rede rodando na maioria das vezes somente aplicação servidora de *Syslog*.

4.2 Resultados

Esses resultados foram coletados em um *honeypot* comprometido e que possuía esta versão modificada do *bash*. As informações de data, hora e número IP apresentadas no resultado foram sanitizadas para assegurar a confidencialidade do mesmo.

```
IP=10.0.0.13 TS=Thu Jul 17 08:54:53 2003 \
UID=1000 PID=48537 CMD=ls
IP=10.0.0.13 TS=Thu Jul 17 08:55:14 2003 \
UID=1000 PID=48537 CMD=du -sh
IP=10.0.0.13 TS=Thu Jul 17 08:56:20 2003 \
UID=1000 PID=48537 CMD=for i in *.[c]
IP=10.0.0.13 TS=Thu Jul 17 08:56:24 2003 \
UID=1000 PID=48537 CMD=do
IP=10.0.0.13 TS=Thu Jul 17 08:56:39 2003 \
UID=1000 PID=48537 CMD=grep include $i
IP=10.0.0.13 TS=Thu Jul 17 08:56:41 2003 \
UID=1000 PID=48537 CMD=done
```

5 RECONSTRUÇÃO DE SESSÃO COM MÓDULOS DE *KERNEL*

Está sendo desenvolvido uma ferramenta, denominada SMaRT(*Session Monitoring and Replay Tool*) para acompanhar as atividades dos invasores à medida que elas acontecem, e para reproduzí-las sempre que for necessário(*offline*). Uma primeira versão desta ferramenta já está sendo testada. A figura 1 mostra o resultado da captura de uma sessão num ambiente de teste.

A captura das atividades está sendo feita através de módulos de *kernel*(LKM) aplicando a técnica de interceptar a função *tty_write* descrita neste artigo. Esses módulos ficam alocados nos *honeypots*, capturando todos os dados dos terminais, transmitindo essas informações para estações de monitoração.

Os dados capturados são cifrados e transferidos para uma máquina remota utilizando uma interface de rede comum, a mesma que é utilizada pelos atacantes para acessar o *honeypot*, porém o módulo *kernel* instalado provê uma funcionalidade de ocultar os dados do canal de comunicação, iludindo os invasores. Como os analisadores de protocolos conseguem analisar tudo o que passa pela rede, as informações referentes a esses dados são ocultadas antes de serem entregues ao usuário [19] [20].

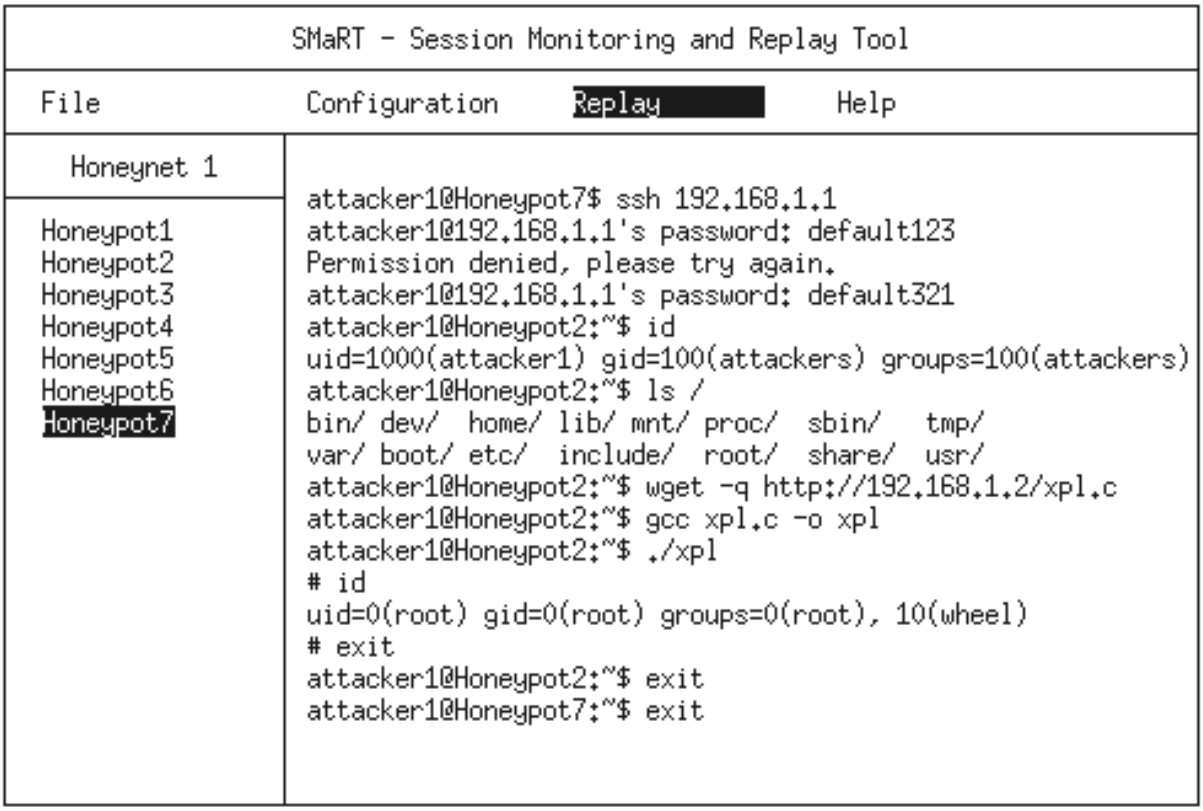

Figura 1: Captura de tela da ferramenta SMaRT reconstruindo uma sessão

5.1 Arquitetura da Solução

Cada *honeypot* conterá um sistema de monitoração(módulos de *kernel*) visando o registro das atividades dos atacantes, e os dados capturados por este sistema serão enviados para uma máquina central.

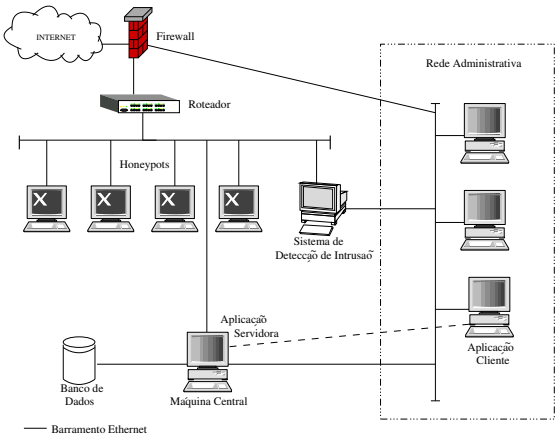

Figura 2: Arquitetura do sistema

A máquina central(estação de monitoração) além de capturar os dados exportados pelos *honeypots*, terá a responsabilidade decifrar, interpretar, armazenar e disponibilizá-los. Caso os administradores da *honeynet* pretendam acompanhar os ataques à medida que eles ocorram, a estação de monitoração, após processar os dados, poderá disponibilizá-los para os interessados. Este acompanhamento será feito com a ajuda de duas aplicações: uma servidora e uma cliente. A aplicação servidora também rodará na máquina central e ficará encarregada de fornecer dados às aplicações clientes que poderão ser instaladas em qualquer máquina que possua permissão de acesso a máquina central.

6 CONCLUSÕES

As técnicas e ferramentas apresentadas neste artigo são de extrema importância na captura das atividades em *honeypots*, visto que a comunidade dos invasores usa frequentemente ferramentas com criptografia para acesso às maquinas comprometidas [9].

Outras técnicas e ferramentas podem ser utilizadas com este propósito, e é o que vem motivando várias instituições ligadas à pesquisa de *honeypots* e *honeynets*.

A monitoração utilizando uma versão modificada do *bash*, ou outros interpretadores de comandos, é uma solução simples, rápida e pode ser aplicada à vários sistemas operacionais. Em contrapartida *rootkits LKM* [13] podem alterar os resultados apresentados por essas técnicas em *user space*.

Já as técnicas baseadas em programação de *ker-*

nel são mais promissoras por poderem gerenciar todas as atividades do sistema operacional. A ferramenta SMaRT está sendo desenvolvida justamente para aproveitar dessas características e monitorar de forma imperceptível todas as atividades dos invasores em *honeypots* de alta interatividade.

AGRADECIMENTOS

Os autores agradecem a ajuda do grupo Honeynet.BR[11] durante a elaboração deste artigo e principalmente pelas contribuições no desenvolvimento do projeto SMaRT.

REFERÊNCIAS

[1] S. Tzu, *The Art of War*. Random House, 1981. ISBN 0-877-73452-6.

[2] C. Stoll, "Stalking the Wily Hacker," *Communications of the ACM*, vol. 31, pp. 484–497, Maio 1988. http://cne.gmu.edu/modules/acmpkp/security/texts/HACKER.PDF.

[3] C. Stoll, *The Cuckoo's Egg: Tracking a Spy Through the Maze of Computer Espionage*. Garden City, NY: Doubleday, 1989. ISBN 0-385-24946-2.

[4] W. R. Cheswick, "An Evening with Berferd in Which a Cracker is Lured, Endured, and Studied," in *Proceedings of the Winter 1992 USENIX Conference*, (San Francisco, California, USA), pp. 163–174, 1992. http://cne.gmu.edu/modules/acmpkp/security/texts/CRACKER.PDF.

[5] S. M. Bellovin, "There Be Dragons," in *Proceedings of the Third Usenix Security Symposium*, 1992. http://www.research.att.com/~smb/papers/dragon.ps.

[6] L. Spitzner, "Honeypots: Definitions and Value of Honeypots," Maio 2002. http://www.spitzner.net/honeypot.html.

[7] L. Spitzner, *Honeypots: Tracking Hackers*. Addison-Wesley, 2003. ISBN 0-321-10895-7.

[8] The Honeynet Project, *Know Your Enemy: Revealing the Security Tools, Tactics, and Motives of the Blackhat Community*. Addison-Wesley, 1st ed., Agosto 2001. ISBN 0-201-74613-1.

[9] A. B. Filho, A. S. M. S. Amaral, A. Montes, C. Hoepers, K. Steding-Jessen, L. H. Franco, and M. H. P. C. Chaves, "Honeynet.BR: Desenvolvimento e Implantação de um Sistema para Avaliação de Atividades Hostis na Internet Brasileira," in *Anais do IV Simpósio sobre Segurança em Informática (SSI'2002)*, (São José dos Campos, SP), pp. 19–25, Novembro 2002. http://www.honeynet.org.br/papers/hnbr-ssi2002.pdf.

[10] A. Silberschatz, G. Gagne, and P. Galvin, *Operating System Concepts*. Student Computing Series, John Wiley & Sons, 1999. ISBN 0-471-36508-4.

[11] J. Corbet and A. Rubini, *Linux Device Drivers*. O'Reilly & Assoc, 2001. ISBN 0-596-00008-1.

[12] S. A. Maxwell, *Linux Core Kernel Commentary*. Coriolis, 2001. ISBN 1-588-80149-7.

[13] N. Murilo and K. Steding-Jessen, "Métodos para Detecção Local de Rootkits e Módulos de Kernel Maliciosos em Sistemas Unix," in *Anais do III Simpósio sobre Segurança em Informática (SSI'2001)*, (São José dos Campos, SP), pp. 133–139, Outubro 2001. http://www.chkrootkit.org/papers/chkrootkit-ssi2001.pdf.

[14] W. R. Stevens, *Advanced Programming in the UNIX Environment*. Addison-Wesley Publishing Company, 1992. ISBN 0-201-56317-7.

[15] rd, "Writing Linux Kernel Keylogger," *Phrack Inc.*, vol. 59, Junho 2002. http://www.phrack.org/show.php?p=59\&a=14.

[16] W. R. Stevens, *TCP/IP Illustrated, Volume 1: The Protocols*. Addison-Wesley, 1998. ISBN 0-201-63346-9.

[17] S. McCanne and V. Jacobson, "The BSD packet filter: A new architecture for user-level packet capture," in *USENIX Winter*, pp. 259–270, 1993. http://citeseer.nj.nec.com/mccanne92bsd.html.

[18] W. R. Stevens, *UNIX Network Programming Volume 1 Networking APIs: Sockets and XTI*. Prentice-Hall, 2 ed., 1998. ISBN 0-13-490012-X.

[19] G. Insolvibile, "Kernel Korner: Inside the Linux Packet Filter." Linux Journal, Vol 94, Fevereiro, 1 2002. http://www.linuxjournal.com/article.php?sid=4852.

[20] G. Insolvibile, "Kernel Korner: Inside the Linux Packet Filter II ." Linux Journal, Vol 95, Março, 1 2002. http://www.linuxjournal.com/article.php?sid=5617.

[11] Amândio Balcão Filho, Antonio Montes, Cristine Hoepers, Klaus Steding-Jessen, Lucio Henrique Franco, Luiz Gustavo C. Barbato e Marcelo H. P. Caetano Chaves