MÉTODOS PARA DETECÇÃO LOCAL DE ROOTKITS E MÓDULOS DE KERNEL MALICIOSOS EM SISTEMAS UNIX

Nelson Murilo
Pangéia Informática
SRTS 701, 70 cj E, sala 304
70340-902 Brasília-DF
nelson@pangeia.com.br

Klaus Steding-Jessen
NIC BR Security Office
Rua Hum, 45 Caixa Postal 6146
13083-970 Campinas-SP
jessen@nic.br

RESUMO

Rootkits são ferramentas utilizadas com freqüência por invasores para ocultar sua presença em máquinas comprometidas, fazendo parte inclusive de alguns Worms e ferramentas de DDoS. Uma invasão pode passar meses sem ser descoberta graças a essas ferramentas. Além das versões tradicionais, rootkits vem sendo implementados também sob a forma de módulos de kernel (LKMs), dificultando a sua detecção. Este artigo descreve o funcionamento desses dois tipos de rootkit, alguns métodos para sua detecção em sistemas Unix e uma ferramenta de código aberto, implementada pelos autores, para detecção automatizada de rootkits.

ABSTRACT

Rootkits are a very popular technique among intruders to hide themselves in compromised machines. Some rootkits are also used by Worms and DDoS tools. An intrusion can remain undetected for months when such a tool is used by an attacker. Rootkits are also being implemented as Loadable Kernel Modules (LKMs), making them much harder to detect. This paper describes the traditional as well as the LKM-based rootkit and presents some rootkit detection methods for Unix machines. This paper also presents an open source tool, developed by the authors, to detect rootkits.

1 INTRODUÇÃO

Com a popularização de ferramentas de ataque automatizado, comprometer máquinas Unix com vulnerabilidades conhecidas tornou-se atividade extremamente simples [1].

Após uma fase inicial de varredura o atacante localiza uma máquina com alguma vulnerabilidade e a explora, obtendo assim acesso privilegiado. Nesse momento a preocupação é não ser detectado e garantir futuros meios de acesso ao sistema através da instalação de *backdoors* [2–5]. Este ciclo eventualmente recomeça com a utilização da máquina invadida para a varredura e invasão de outros sistemas vulneráveis.

O procedimento típico de invasão de uma máquina Unix pode ser dividido genericamente em 4 passos básicos, como pode ser visto no diagrama da Fig. 1.

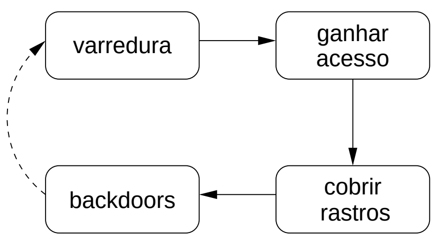

Figura 1: Fases básicas de uma invasão.

A instalação de rootkits é uma maneira bastante utilizada pelos invasores para não serem detectados no sistema e instalar *backdoors*.

O termo rootkit refere-se a um conjunto de ferramentas usadas pelo invasor não para obter privilégios de root, mas sim para manter esses privilégios [6].

Embora existam rootkits também para a plataforma Windows [3, 7], este artigo concentra-se apenas em rootkits para sistemas Unix.

O artigo inicialmente apresenta um histórico da evolução dos rootkits, seguido da descrição dos componentes e funcionamento dos rootkits tradicionais e também dos implementados sob a forma de LKMs. São discutidos em seguida alguns métodos de detecção para esses tipos de rootkit. Por fim é mostrada uma ferramenta gratuita e de código aberto, desenvolvida pelos autores, para detecção automatizada de rootkits.

2 HISTÓRICO

A partir de 1989 começaram a surgir, na revista "underground" Phrack Magazine, códigos-fonte de ferramentas que permitiam editar e remover entradas nos arquivos `utmp`, `wtmp` e `lastlog`[1] do sistema. O objetivo dessas ferramentas era ocultar um invasor numa máquina Unix comprometida—desse modo sua presença não era revelada por comandos como `w`, `who`, `users` e `last` [8–10].

Com o tempo foram sendo desenvolvidas várias outras ferramentas com o objetivo de ocultar um invasor numa máquina Unix: versões de comandos do sistema que ocultam informações relativas ao invasor, programas para não alterar os tempos de acesso, modificação e o CRC de programas alterados, etc. Estas ferramentas reunidas deram origem aos primeiros rootkits.

Os primeiros rootkits, originalmente escritos para SunOS e depois para Linux, são citados, em conjunto com o uso de *sniffers*, em *advisories* do CERT já em 1994 [11, 12].

Em 1995 torna-se muito popular na comunidade underground um rootkit denominado "RootKit", usado num grande número de invasões [13]. A partir daí

[1] Em sistemas Unix, o arquivo `utmp` mantém a lista dos usuários ativos, o `wtmp` a lista de *logins* e *logouts* e o arquivo `lastlog` o último *login* de cada usuário.

muitos outros tipos de rootkits apareceram, para praticamente todas as variantes de Unix.

A partir de 1997 surgem referências de construção de rootkits inseridos diretamente no *kernel*, sob forma de módulos de kernel (*Loadable Kernel Modules*, ou LKMs) [14–16].

Nos dias atuais rootkits continuam sendo muito populares entre invasores, fazendo parte inclusive de alguns *worms* [17, 18] e ferramentas de DDoS [19].

3 O ROOTKIT TRADICIONAL

Por rootkit tradicional entende-se o rootkit que não é implementado em *kernel space* sob a forma de LKM. Tipicamente esse tipo de rootkit é composto, pelo menos, de:

- versões modificadas de comandos importantes do sistema como `ls`, `ps`, `netstat`, `crontab`, `tcpd`, `syslogd`, etc. Os comandos substituídos podem omitir processos, conexões, arquivos e *logs* do invasor, de modo que sua presença não seja notada pelo administrador e demais usuários do sistema.

Geralmente contém algumas ferramentas adicionais:

- Programas para remoção de entradas em arquivos de *log* do sistema bem como nos arquivos `utmp`, `wtmp` e `lastlog`, como por exemplo `z2` e `wted`.
- Ferramentas para alterar os tempos de modificação, acesso e *status* (*MAC times*), o tamanho e o CRC[2] dos arquivos modificados pelo invasor, de modo a torná-los iguais aos originais. (por exemplo as ferramentas `addlen` e `fix`).
- *Sniffers* para a captura de senhas que trafeguem pela rede sem criptografia, como por exemplo em protocolos como `ftp`, `telnet`, `pop2`, `pop3`, `imap2`, `rlogin`, etc. Nesse caso geralmente o comando `ifconfig` é modificado para que não indique que uma determinada interface de rede está em modo promíscuo [6, 20].

  Modificações em clientes `ssh` também se tornaram um excelente método usado por invasores para obtenção de senhas.
- *Backdoors* que garantam o acesso remoto futuro à máquina, mesmo em caso de mudança de senhas ou correções de vulnerabilidades. Geralmente permitem acesso privilegiado (root) e não deixam registro nos *logs* do sistema.

  Esse acesso remoto pode ser obtido através da instalação de um processo que escuta em uma determinada porta e inicia uma *shell* (por exemplo o programa `bindshell`). Uma chamada a esse tipo de processo é comumente inserida nos *scripts* de inicialização da máquina para garantir seu funcionamento mesmo por ocasião da reinicialização do sistema. Alguns *backdoors* desse tipo implementam autenticação com senha e sessões criptografadas.

  Outro mecanismo que o invasor possui para garantir acesso remoto é a modificação de *daemons* do sistema, como `inetd`, e `sshd`, entre outros. Embora pareça estar funcionando normalmente, a versão modificada usualmente é programada para escutar em alguma porta alternativa que inicia uma *shell* ou aceitar alguma senha especial que garanta acesso privilegiado.
- Ferramentas para o aumento de privilégios de contas locais.

  Caso o invasor perca o acesso remoto privilegiado através de *backdoors* ou escolha entrar usando uma conta do sistema, mecanismos que permitam a obtenção de privilégio de root através de uma conta comum são geralmente utilizados.

  Para permitir esse aumento de privilégio alguns comandos são alterados e fornecem uma *shell* de root ao serem executados de modo especial, como por exemplo através de uma senha. Exemplos de comandos normalmente alterados são `chfn` e `chsh`.
- *Exploits*, programas de *Denial of Service* (DoS) e *Scanners*.

  É muito comum uma máquina invadida estar sendo usada apenas como base para comprometer outras redes—desse modo muitos rootkits já incluem ferramentas para realizar as fases de varredura e exploração de vulnerabilidades em outras máquinas, como descrito na Fig. 1.
- Programas relacionados com IRC, como *bouncers*.

  Estes programas permitem que o invasor se conecte em canais de IRC usando a máquina invadida como "*proxy*", desse modo não revelando o seu ponto de origem real.

4 O LKM ROOTKIT

Módulos de *kernel* (*Loadable Kernel Modules* ou LKMs) são componentes que podem ser dinamicamente carregados no *kernel*, modificando a funcionalidade de um sistema sem a necessidade de uma reinicialização. LKMs são implementados em vários sistemas Unix (Linux, BSD, Solaris, HP-UX) e geralmente usados para carregar *device drivers* de maneira dinâmica.

[2] Para fins de checagem de integridade o uso de CRCs (de 16 e 32 bits) não é recomendado. Um invasor pode facilmente gerar versões modificadas de arquivos que possuam os mesmos CRCs dos originais, quando verificados pelo comando `sum` do Unix. Para um resultado mais confiável, o uso de algum hash criptográfico, como MD5 ou SHA-1, é recomendável.

Enquanto os rootkits tradicionais funcionam alterando os comandos do sistema para omitir informações da presença do invasor, como conexões, arquivos, processos, etc, os rootkits implementados sob forma de LKMs funcionam alterando chamadas do sistema (*syscalls*) tais como `kill`, `getdents` e `read` e atuando diretamente em algumas estruturas do *kernel* [16].

Do ponto de vista do administrador a detecção desses rootkits implementados como LKMs é muito mais difícil, pois todos os comandos do sistema continuam inalterados e o próprio *kernel* responderá às requisições mas ocultando a presença do invasor. Mesmo uma checagem dos *hashes* criptográficos de todos os comandos do sistema não apontará nenhuma modificação. Normalmente esse tipo de rootkit altera também as chamadas do sistema que permitem listar os módulos de *kernel* instalados.

5 WORMS

Worms são programas com a capacidade de comprometer outras máquinas e se propagar para elas de modo automático usando a rede. Tipicamente executam uma varredura por sistemas e serviços vulneráveis, exploram essas vulnerabilidades e se copiam para a máquina recém explorada. Uma vez instalados na nova máquina, o ciclo à procura de novas máquinas recomeça.

Outras ações em alguns casos incluem: instalação de rootkits para evitar sua detecção, correção da vulnerabilidade explorada para evitar ataques por terceiros, envio de email com informações sobre o sistema, instalação de *backdoors*, *web defacements*, etc. [17, 18, 21–23]

6 MÉTODOS DE DETECÇÃO

A seguir são comentados alguns métodos para detecção de rootkits tradicionais, rootkits implementados em módulos de *kernel* (LKMs) e *Worms*:

6.1 Busca por Arquivos de Configuração

Muitos componentes de rootkits consultam, durante sua execução, arquivos de configuração. Nesses arquivos o invasor normalmente insere quais informações devem ser ocultadas do usuário, como processos, arquivos, conexões e entradas em arquivos de *log* [20].

A Fig. 2 mostra um trecho do arquivo de cabeçalho usado no Linux Rootkit 6 (lrk6) que define, em tempo de compilação, a senha e os arquivos de configuração usados pelos vários programas que compõe este rootkit.

É possível localizar a presença de um rootkit procurando por ocorrências de nomes de arquivos de configuração no interior de comandos alterados por um rootkit. As ferramentas `strings`, `od` e `hexdump` do Unix são normalmente usadas para esse tipo de análise.

```
/* LINUX ROOTKIT DEFINES. */

#define PW_LEN  6

#define ROOTKIT_PASSWORD "vejeta"

/* Processes to hide */
#define ROOTKIT_PROCESS_FILE "/dev/hda01"

/* Addresses to hide */
#define ROOTKIT_ADDRESS_FILE "/dev/hda02"

/* Files and directories to hide */
#define ROOTKIT_FILES_FILE "/dev/hda03"

/* Log entries to hide */
#define ROOTKIT_LOG_FILE "/dev/hda04"

#define ROOTKIT_IFE_FILE "/dev/hda05"

#define ROOTKIT_TELNET_FILE "/dev/hda06"
```

Figura 2: Trecho do `rootkit.h` do lrk6.

6.2 Busca por Seqüências de Caracteres em Arquivos

Além de arquivos de configuração, outras evidências de comprometimento podem ser obtidas através do exame de cadeias de caracteres em comandos do sistema modificados por rootkits. Essas seqüências incluem senhas, endereços de email, a tabela de símbolos, compilador e bibliotecas usadas na compilação.

6.3 Comparação de Hashes Criptográficos

Hashes criptográficos como MD5 e SHA-1 podem ser úteis na determinação de quais comandos do sistema foram alterados por rootkis. Comparando-se o *hash* de um comando com suspeita de alteração com o *hash* do mesmo comando feito no momento da instalação da máquina (ou de outra mídia confiável, como um CDROM de distribuição do sistema) é possível identificar uma modificação.

Esse método exige que os *hashes* dos comandos a serem testados sejam conhecidos e que tenham sido obtidos de binários confiáveis. Exige também que esses *hashes* de comparação sejam mantidos de forma segura, de maneira a não poderem ser alterados pelo invasor, e que o próprio programa que calcula os *hashes* não tenha sido comprometido.

Alguns sistemas possuem um esquema de *packages* que permite a verificação dos comandos instalados no sistema com uma base de *hashes* (geralmente MD5) mantida *on-line*.

6.4 Exame de Chamadas do Sistema

Outro método de detecção é o exame das chamadas do sistema efetuadas por um programa sob suspeita de modificação e observar se existem acessos a arquivos de configuração de rootkits.

Os comandos `strace` (Linux), `ktrace` / `kdump` (BSD) e `truss` (Solaris) podem ser usados para este fim [24].

6.5 *Exame dos Tempos de Modificação, Acesso e Status (MAC Times)*

Para cada arquivo e diretório do sistema de arquivos do Unix estão associados três atributos de tempo: última modificação (`mtime`), último acesso (`atime`) e última mudança no *inode* (`ctime`).

Através do exame cuidadoso desses atributos em um determinado sistema é possível achar evidências de uma invasão, da instalação de um rootkit por um invasor ou da presença de um *worm* [25, 26].

6.6 *Exame de Conexões*

Vários rootkits e *worms* instalam *backdoors* escutando em diversas portas da máquina. Descobrir essas portas em estado de `LISTEN`, bem como conexões suspeitas, pode servir para localizar os processos associados a essas portas de *backdoor*.

Normalmente o comando `netstat` é alterado para não mostrar tais informações. No caso de LKMs estas são ocultadas diretamente pelo *kernel* e uma varredura a partir de uma máquina remota pode se fazer necessária.

6.7 *Exame da Tabela de Chamadas do Sistema e Símbolos do Kernel*

O exame da tabela de chamadas do sistema, usando por exemplo uma ferramenta como o `kstat` em sistemas Linux, pode ser extremamente útil para se determinar a presença de um LKM rootkit [27].

Alguns LKM rootkits deixam definições de símbolos característicos que podem ser observados na tabela de Símbolos do *kernel* (`/dev/ksyms` em alguns sistemas).

6.8 *Busca por Outros Indícios de Comprometimento*

Muitas vezes os indícios de um rootkit não são encontrados facilmente. Nesses casos é extremamente importante que o administrador seja capaz de reconhecer alguns indícios clássicos de invasão e assim intensificar suas buscas pelo rootkit.

Evidências de invasão incluem: interfaces de rede em modo promíscuo (instalação de *sniffer*), evidências de remoção de entradas nos arquivos `utmp`, `wtmp` e `lastlog` do sistema, arquivos regulares abaixo de `/dev` (este diretório contém normalmente apenas arquivos de dispositivos), arquivos e diretórios começando com '`.`' ou espaço, entre outras. [24, 28]

7 IMPLEMENTAÇÃO

Nem todos os administradores de sistemas têm o conhecimento necessário para por em prática todos os métodos apresentados acima. Mesmo um administrador experiente pode se beneficiar automatizando alguns dos testes, principalmente se for responsável por um grande número de máquinas.

Embora já existam algumas ferramentas para deteção de rootkits, seus testes são restritos a poucos sistemas e comandos [29, 30] ou específicos para LKM rootkits [31, 32].

Considerando esses fatores os autores desenvolveram uma ferramenta gratuita e de código aberto que implementa grande parte dos métodos de detecção de rootkits expostos neste artigo.

7.1 *Decisões de Implementação*

Dada a grande evolução dos rootkits, a crescente facilidade para efetuar invasões e o grande número de plataformas para as quais existem rootkits, chegou-se à conclusão de que para ser útil a ferramenta de detecção deveria rodar no maior número possível de sistemas Unix. Isso inclui sistemas que não possuem compiladores C nem interpretadores como Perl em sua instalação padrão. Decidiu-se então escrever a base do programa em *Bourne Shell*, por ser padrão dentre os diversos sistemas Unix.

O programa principal, `chkrootkit`, faz a grande maioria dos testes, usando comandos padrão do Unix e invocando alguns programas auxiliares escritos em C. Caso esses programas auxiliares não estejam disponíveis (por não haver um compilador C instalado) os demais testes ainda podem ser executados.

7.2 *Componentes*

Um sumário dos componentes da ferramenta, suas funcionalidades e tamanhos (em linhas de código), pode ser observado na Tab. 1.

Tabela 1: Componentes do `chkrootkit` (versão 0.34).

Componente	Linhas	Descrição
`chkrootkit`	1985	*Shell Script* que contém as assinaturas dos rootkits conhecidos e executa alguns testes externos.
`chklastlog.c`	282	Verifica remoções no arquivo `lastlog`
`chkwtmp.c`	90	Verifica remoções no arquivo `wtmp`
`ifpromisc.c`	141	Testa por interfaces em modo promíscuo
`chkproc.c`	96	Testa por sinais de LKMs genéricos

7.2.1 *chkrootkit*

Shell script que implementa os testes principais e invoca, opcionalmente, os demais programas da ferramenta.

Inicialmente o *script* testa se os seguintes comandos, usados nos testes para a detecção de rootkits, estão presentes na máquina: `awk`, `cut`, `echo`, `egrep`, `find`, `head`, `id`, `ls`, `netstat`, `ps`, `sed`, `strings` e `uname`.

Como os comandos citados acima também podem estar modificados por algum rootkit e assim comprometer a detecção, o *script* prevê duas outras possibilidades de operação:

1. O uso de um *path* alternativo para os comandos acima, através da opção de linha de comando '`-p`'. Com essa opção definem-se outro(s) diretório(s) onde o *script* deve buscar seus comandos. Essa opção permite ao administrador, por exemplo, o uso de comandos de uma mídia confiável, como um CDROM ou *floppy*, no exame de uma máquina com suspeita de instalação de rootkits.

2. Especificar um diretório raiz alternativo, através da opção '`-r`'. Desse modo é possível fazer uma análise *post-mortem* de um sistema levando seu(s) disco(s) para outra máquina e executando o `chkrootkit` a partir dela. Esse método tem como vantagem o fato de poder contar, além de binários, com um *kernel* confiável, eliminando assim a possibilidade de um LKM na máquina que se deseja examinar ocultar dados importantes.

Se todos os comandos necessários estão presentes no sistema o *script* começa os testes. O comportamento padrão é realizar todos os testes disponíveis, mas o usuário pode especificar, na linha de comando, apenas os testes de interesse.

Existem duas categorias de testes: internos, isto é, realizados pelo próprio *script* e os testes externos, realizados por programas em C que acompanham a ferramenta.

Os testes internos são implementados como funções em *Bourne Shell*. Um exemplo de uma função desse tipo é mostrada na Fig. 3.

```
chk_su () {
    STATUS=${NOT_INFECTED}
    SU_INFECTED_LABEL="satori|vejeta"
    CMD=`loc su su $pth`

    if [ "${EXPERT}" = "t" ]; then
        expertmode_output "${strings} -a ${CMD}"
        return 5
    fi

    if ${strings} -a ${CMD} | ${egrep} \
       "${SU_INFECTED_LABEL}" > /dev/null 2>&1
    then
        STATUS=${INFECTED}
    fi
    return ${STATUS}
}
```

Figura 3: Função `chk_su()`, implementada pelo *script* `chkrootkit`. Esta função testa a ocorrência de duas cadeias associadas à modificação por rootkits (senhas padrão do rootkit lrk3 e lrk6) no comando `su`. A linha do `if` foi quebrada por questões de legibilidade.

Essas funções geralmente iniciam tentando localizar o comando a ser testado, visto que muitas vezes a localização de um comando difere bastante entre os vários sistemas Unix. Do mesmo modo, alguns testes são efetuados de maneira diferente dependendo do sistema no qual o *script* está executando.

Muitos testes procuram por assinaturas específicas dentro do comando que está sendo testado, isto é, ocorrência de cadeias de caracteres que são sabidamente associadas a rootkits.

Outros testes incluem checagem por permissões não usuais, nomes de arquivos e diretórios associados a rootkits e *worms* conhecidos, indícios que o arquivo de "history" do usuário root tenha sido apagado, etc.

A determinação das cadeias de caracteres e nomes de arquivos e diretórios provém da análise de rootkits conhecidos e de informações obtidas através da interação com a comunidade de segurança.

Implementou-se também um modo de operação da ferramenta denominado *expert mode*. Nesse modo toda a interpretação da saída dos comandos executados é deixada a cargo do usuário. É útil para a detecção de variantes de rootkits que ainda não possuem assinaturas incorporadas ao programa.

Atualmente são implementados 57 testes para determinar a presença de rootkits tradicionais, LKMs e *worms*[3].

7.2.2 chklastlog

Esta ferramenta verifica indícios de remoções de entradas do arquivo `lastlog` do sistema.

O programa percorre as entradas do arquivo `wtmp` e para cada nome de usuário encontrado, testa sua presença no arquivo `lastlog`. Se este usuário não é encontrado emite um alerta, indicando que esta entrada pode ter sido apagada por um invasor. O comando retorna o número de anomalias encontradas.

Foi originalmente escrito pelo DFN-CERT para SunOS e depois adaptado para pelos autores para que pudesse ser utilizado também em Linux, FreeBSD, OpenBSD e Solaris. Outra modificação foi a possibilidade de especificar arquivos de `wtmp` e `lastlog` alternativos através da opção '`-f`'.

7.2.3 chkwtmp

Ferramenta que verifica indícios de remoções de entradas do arquivo `wtmp` do sistema e retorna o número de anomalias encontradas.

O arquivo `wtmp` é aberto e o conteúdo de cada entrada é testado—se for zero emite um alerta. Este teste se baseia no fato de várias ferramentas de edição deste arquivo, como `wted` e `z2`, sobrescreverem as entradas com zeros.

Este programa também foi originalmente escrito pelo DFN-CERT para SunOS e adaptado pelos autores, que o portaram para outros Unices e adicionaram alguma funcionalidade, por exemplo a habilidade de especificar outros arquivos de `wtmp`, não apenas o arquivo padrão do sistema.

[3] Isso refere-se à versão 0.34 do `chkrootkit`.

7.2.4 *ifpromisc*

Este programa testa por interfaces de rede em modo promíscuo, estado geralmente associado à instalação de *sniffers* em máquinas comprometidas.

Obtém do *kernel* a lista de interfaces de rede da máquina, e para cada interface (com excessão da inteface de *loopback*), testa suas *flags* pela condição `IFF_PROMISC`.

Esta ferramenta pode dar falsos positivos em casos de uso legítimo de ferramentas que deixam a interface de rede em modo promíscuo, como por exemplo `tcpdump` e `snort`.

7.2.5 *chkproc*

Alguns sistemas Unix implementam o *process filesystem*, geralmente abaixo de `/proc`. Este *filesystem* contém, entre outras informações, uma entrada para cada processo ativo no sistema. Essas entradas são visíveis como diretórios, onde o nome de cada diretório é o *Process ID* (PID) do processo.

A chamada do sistema `readdir` é normalmente usada para obter-se o conteúdo de um diretório. Alguns LKMs modificam esta chamada de modo a ocultar diretórios abaixo de `/proc` e desse modo ocultar processos do sistema [16].

Embora ocultos da chamada `readdir`, é possível entrar nesses diretórios, com a chamada `chdir`, se o nome do diretório for conhecido.

O `chkproc` tenta, seqüencialmente, um `chdir` em cada um dos diretórios que representam os PIDs de processos do sistema (de 1 a 65535)—toda vez que o `chdir` é bem sucedido o PID associado a este diretório é colocado em uma lista. Estes PIDs são comparados com os resultados obtidos através do uso da chamada `readdir` e com a saída do comando `ps`. Diferenças nesses resultados podem revelar, respectivamente, a presença de um LKM malicioso ou a modificação do comando `ps` com o objetivo de ocultar processos.

Em sistemas onde existe a criação de um grande número de processos de curta duração este comando pode reportar alguns falsos positivos.

7.3 *Exemplos de Uso*

O `chkrootkit` é uma ferramenta de linha de comando e deve ser executada como root:

```
# ./chkrootkit
```

Também é possível fazer testes específicos como nos exemplos a seguir.

Teste pela presença de versões modificadas de `ps` e `ls` e se a interface de rede está em modo promíscuo:

```
# ./chkrootkit ps ls sniffer
```

Especificação de um *path* alternativo para binários confiáveis a serem usados pelo programa.

```
# ./chkrootkit -p /floppy
```

Verificação do disco de uma máquina montado abaixo de `/mnt`:

```
# ./chkrootkit -r /mnt
```

Execução em *expert mode* procurando por *pathnames*:

```
# ./chkrootkit -x | egrep '^/'
```

7.4 *Trabalhos Futuros*

Atualmente todas as assinaturas de rootkits testados são mantidas dentro do corpo do programa. Adições de novos testes e assinaturas implica na alteração do código. Planeja-se separar completamente o código propriamente dito da base de dados de assinaturas, tornando a sua manutenção mais fácil.

Outra meta é consultar uma base de assinaturas de rootkits *on-line*—desse modo a ferramenta poderia obter e manter atualizada a sua base de assinaturas de rootkits através da rede. O projeto de criação de um repositório *on-line* de assinaturas de rootkits, com contribuições da comunidade mundial de segurança, já existe por parte do projeto `cyberabuse.org`. Este repositório poderá ser usado pelo `chkrootkit`.

8 CONCLUSÕES

Rootkits são ferramentas muito poderosas e sua detecção está ficando cada vez mais difícil, principalmente no caso de rootkits implementados como módulos de *kernel*. *Worms* estão ficando comuns e é de se esperar que cada vez mais incorporem rootkits para evitar sua detecção nos sistemas atingidos.

Claramente o crescimento da variedade de rootkits, tanto tradicionais como em módulos de *kernel*, está sendo mais rápido que a capacidade dos administradores de acompanhar essa evolução, dificultando assim a sua detecção.

É extremamente importante que o conhecimento acumulado da comunidade de segurança na detecção de rootkits possa ser convertido na construção de ferramentas automatizadas. Além de auxiliar um grande número de administradores vítimas de invasões, ferramentas como o `chkrootkit` podem tornar o exame periódico de ambientes com muitas máquinas mais fácil e eficiente, auxiliando na detecção precoce de invasões.

AGRADECIMENTOS

Os autores gostariam de agradecer a ajuda de Cristine Hoepers na revisão deste artigo e por sua contribuição ao longo do desenvolvimento do `chkrootkit`, especialmente as idéias e os testes que deram origem ao comando `chkproc`.

REFERÊNCIAS

[1] L. Spitzner, "Know Your Enemy: The Tools and Methodologies of the Script Kiddie." http://project.honeynet.org/papers/enemy/, 2000.

[2] S. Garfinkel and G. Spafford, *Practical UNIX and Internet Security*. O'Reilly & Associates, Inc., second ed., 1996.

[3] J. Scambray, S. McClure, and G. Kurtz, *Hacking Exposed: Network Security Secrets & Solutions*. Osborne/McGraw-Hill, second edition ed., 2001.

[4] L. Spitzner, "Know Your Enemy: III, They Gain Root." http://project.honeynet.org/papers/enemy3/, 2000.

[5] "The Honeynet Project's Forensic Challenge." http://project.honeynet.org/challenge/, January 2001.

[6] D. Brumley, "invisible intruders: rootkits in practice," *;login: Magazine, Intrusion Detection Special Issue*, September 1999. http://www.usenix.org/publications/login/1999-9/features/rootkits.html.

[7] G. Hoglund, "A Real NT Rootkit, Patching the NT Kernel," *Phrack Magazine, Volume 9, Issue 55, File 5*, September 1999. http://www.phrack.org/show.php?p=55&a=5.

[8] B. T. Affair, "Hiding Out Under Unix," *Phrack Magazine, Issue 25, File 6*, March 1989. http://www.phrack.org/show.php?p=25&a=6.

[9] S. Hackalot, "C UNIX nasties part I," *Phrack Magazine, Volume 3, Issue 32, File 5*, November 1990. http://www.phrack.org/show.php?p=32&a=5.

[10] P. Accident, "Playing Hide and Seek, Unix Style," *Phrack Magazine, Issue 43, File 14*, Jul 1993. http://www.phrack.org/show.php?p=43&a=14.

[11] CERT/CC, "CERT Advisory CA-94-01 Ongoing Network Monitoring Attacks." http://www.cert.org/advisories/CA-1994-01.html, 1994.

[12] CERT/CC, "CERT Advisory CA-95-18 Widespread Attacks on Internet Sites." http://www.cert.org/advisories/CA-1995-18.html, December 1995.

[13] D. H. Freedman and C. C. Mann, *At large: the strange case of the world's biggest internet invasion*. Simon & Schuster, 1997. pg. 282–283.

[14] halflife, "Abuse of the Linux Kernel for Fun and Profit," *Phrack Magazine, Volume 7, Issue 50, File 5*, Apr 1997. http://www.phrack.org/show.php?p=50&a=5.

[15] plaguez, "Weakening the Linux Kernel," *Phrack Magazine, Volume 8, Issue 52, File 18*, Jan 1998. http://www.phrack.org/show.php?p=52&a=18.

[16] pragmatic, "Complete Linux Loadable Kernel Modules." http://packetstorm.securify.com/docs/hack/LKM_HACKING.html, 1999.

[17] M. Vision, "Lion internet worm analysis." http://www.whitehats.com/library/worms/lion/index.html, 2001.

[18] M. Fearnow and W. Stearns, "Adore Worm." http://www.sans.org/y2k/adore.htm, April 2001.

[19] R. Wash and J. Nazario, "Analysis of a Shaft Node and Master." http://biocserver.cwru.edu/~jose/shaft_analysis/, 2000.

[20] D. Dittrich, "Rootkits and hiding files / directories / processes after a break-in." http://staff.washington.edu/dittrich/misc/faqs/rootkits.faq, 2001.

[21] CERT/CC, "CERT Incident Note IN-2001-05 The "cheese" Worm." http://www.cert.org/incident_notes/IN-2001-05.html, May 2001.

[22] CERT/CC, "CERT Incident Note IN-2001-01 Widespread Compromises via "ramen" Toolkit." http://www.cert.org/incident_notes/IN-2001-01.html, January 2001.

[23] CERT/CC, "CERT Advisory CA-2001-11 sadmind/IIS Worm." http://www.cert.org/advisories/CA-2001-11.html, May 2001.

[24] D. Dittrich, "Basic Steps in Forensic Analysis of Unix Systems." http://staff.washington.edu/dittrich/misc/forensics/, 2000.

[25] D. Farmer, "What Are MACtimes?," *Dr. Dobb's Journal*, October 2000. http://www.ddj.com/articles/2000/0010/0010f/0010f.htm.

[26] W. Venema and D. Farmer, "The Coroner's Toolkit." http://www.fish.com/forensics/.

[27] T. Miller, "Detecting Loadable Kernel Modules (LKM)." http://members.prestige.net/tmiller12/papers/lkm.htm, 2001.

[28] CERT/CC, "CERT/CC Intruder Detection Checklist." http://www.cert.org/tech_tips/intruder_detection_checklist.html.

[29] D. Simpson, "checkps: Linux rootkit detector." http://sourceforge.net/projects/checkps/.

[30] A. Daviel, "rkdet: rootkit detector for Linux." http://vancouver-webpages.com/rkdet/.

[31] S. Aubert, "Rkscan: Rootkit Scanner for Loadable Kernel-module Rootkits." http://www.hsc.fr/ressources/outils/rkscan/index.html.en.

[32] K. Mandia and K. J. Jones, "Carbonite: A Linux Kernel Module to aid in RootKit detection." http://www.incident-response.org/Carbonite.htm.