

## Detectando Intrusões na Máquina Virtual *User-Mode Linux*

Marcos Laureano, Carlos Maziero, Edgard Jamhour

Programa de Pós-Graduação em Informática Aplicada  
Pontifícia Universidade Católica do Paraná  
80.215-901 Curitiba – PR, Brasil

{laureano,maziero,jamhour}@ppgia.pucpr.br

**Resumo:** *Os sistemas de detecção de intrusão monitoram continuamente a atividade de um sistema ou rede, buscando evidências de intrusões. Entretanto, detectores de intrusão baseados em host podem ser adulterados ou desativados por invasores bem sucedidos. Este trabalho apresenta uma arquitetura que visa proteger detectores de intrusão baseados em host através de máquinas virtuais. A proposta aqui apresentada usa o isolamento de espaços de execução provido por um ambiente de máquinas virtuais para separar o detector de intrusão do sistema a monitorar. Em consequência, o detector de intrusão torna-se invisível e inacessível a eventuais intrusos. Os testes realizados mostram a viabilidade dessa solução.*

**Palavras-chave:** *detecção de intrusão, máquinas virtuais, segurança, sistemas operacionais de rede.*

**Abstract:** *Intrusion detection systems continuously watch the activity of a network or system, looking for intrusion evidences. However, host-based intrusion detectors may be tampered or disabled by successful intruders. This work presents an architecture aimed to protect intrusion detectors by means of virtual machines. The proposal presented here uses the execution space isolation provided by a virtual machine environment to separate the intrusion detector from the system to monitor. Consequently, the intrusion detector becomes invisible and inaccessible to intruders. The evaluation tests showed the viability of the proposal.*

**Keywords:** *intrusion detection, virtual machines, operating systems, security.*

### 1. Introdução

Diversas ferramentas contribuem para aumentar a segurança de um sistema de computação, entre as quais se destacam os sistemas de detecção de intrusão. Tais sistemas monitoram continuamente a atividade de um ambiente computacional, buscando evidências de intrusões.

Máquinas virtuais podem ser usadas para melhorar a segurança de um sistema computacional contra ataques a seus serviços [Chen 2001]. O conceito de máquina virtual foi definido nos anos 60; no ambiente IBM VM/370, uma máquina virtual propiciava um ambiente de execução exclusivo para cada usuário [Goldberg 1973]. O uso de máquinas virtuais vem se tornando interessante também em sistemas computacionais modernos, devido a suas vantagens em termos de custo e portabilidade [Blunden 2002]. Exemplos de ambientes de máquinas virtuais de uso corrente incluem o sistema VMWare [VMWare 1999] e o *UML – User-Mode Linux* [Dike 2000]. Um uso frequente de sistemas baseados em máquinas virtuais é a chamada *consolidação de servidores*: ao invés de usar vários equipamentos fisicamente separados, faz-se uso de apenas um equipamento, sobre o qual várias

máquinas virtuais isoladas entre si hospedam diferentes sistemas operacionais convidados, com suas aplicações e serviços.

Este trabalho apresenta a proposta e implementação de uma arquitetura destinada a proteger detectores de intrusão baseados em *host*, através da aplicação do conceito de máquina virtual. A proposta faz uso do isolamento de espaços de execução provido por um ambiente de máquinas virtuais para separar o detector de intrusão do sistema a monitorar. Essa separação protege o detector de intrusão, tornando-o invisível e inacessível aos processos dos sistemas operacionais convidados (e, portanto, a eventuais intrusos).

## 2. Detecção de intrusão

Um sistema de detecção de intrusão (IDS – *Intrusion Detection System*) continuamente coleta e analisa dados de um ambiente computacional, buscando detectar ações intrusivas. Em relação à origem dos dados analisados, existem duas abordagens básicas para a detecção de intrusão [Allen 1999]: detectores baseados *em rede* (NIDS – *network-based IDS*), que analisam o tráfego de rede dos sistemas monitorados, e detectores baseados *em host* (HIDS – *host-based IDS*), que monitoram a atividade local em um *host*, como processos, conexões de rede, chamadas de sistema, arquivos de log, etc. A maior deficiência dos detectores baseados em *host* é sua relativa fragilidade: para poder coletar dados da atividade do sistema, o detector (ou um agente do mesmo) deve ser instalado na máquina a monitorar. Esse software pode ser desativado ou modificado por um intruso bem sucedido, para esconder sua presença e suas atividades.

As técnicas usadas para analisar os dados coletados buscando detectar intrusões podem ser classificadas em dois grupos: *detecção de assinatura*, quando os dados coletados são comparados com uma base de padrões (assinaturas) de ataques conhecidos, e *detecção de anomalia*, quando os dados coletados são comparados com dados previamente obtidos sobre o comportamento normal do sistema. Desvios da normalidade são então tratados como ameaças. Uma forma de detectar intrusões é através da análise de seqüências de chamadas de sistema executadas pelos processos, pois estas constituem uma rica fonte de informação sobre a atividade de um sistema. Vários trabalhos científicos descrevem técnicas para detecção de anomalias usando esse tipo de informação. Nessa proposta foi utilizado o algoritmo apresentado em [Forrest 1996, Hofmeyr 1998].

## 3. Alterações Propostas

Através de modificações no monitor de máquinas virtuais é possível coletar, de forma transparente, informações sobre a atividade de cada sistema operacional convidado, incluindo seus usuários, processos, conexões de rede, etc. Essa informação é então enviada para um IDS externo ao ambiente, executando fora do sistema operacional convidado. Fazendo uso de dados previamente armazenados sobre o comportamento do sistema (criados a partir de execuções anteriores), o IDS pode detectar comportamentos anômalos em usuários e/ou processos, ou detectar indícios claros de atividades suspeitas. Se há suspeita de uma intrusão, um sistema de resposta pode agir de forma a impedi-la ou neutralizá-la. Essa funcionalidade pode ser facilmente implementada interceptando as chamadas de sistema emitidas pelos processos do sistema operacional convidado.

## 4. Implementação e resultados

Um protótipo da arquitetura proposta foi implementado em uma plataforma Linux, usando o monitor de máquinas virtuais *UML – User-Mode Linux* [Dike 2000]. O UML foi modificado para permitir extrair dados detalhados do sistema convidado, como as chamadas de sistema emitidas pelos processos convidados. A comunicação entre o monitor UML e os processos externos foi implementada por *pipes* nomeados (desta forma o sistema operacional hospedeiro controla o fluxo de dados entre o monitor e os demais módulos). O hardware usado nos experimentos foi um sistema PC com CPU

Athlon XP 1600 e 512 MBytes de RAM. O sistema hospedeiro usou o *Suse Linux Professional 9.0* (kernel 2.4.21), e como sistema convidado foi usado um *Debian GNU/Linux 2.2* com kernel 2.6.1.

O sistema proposto tem dois modos de operação: um modo de *aprendizagem* e um modo de *monitoração*. Quando operando no modo de aprendizagem, todos os processos executando no sistema convidado e seus respectivos usuários são registrados como usuários e processos autorizados, assim como as seqüências de chamadas de sistema. Quando operando no modo monitoração, o módulo de detecção de intrusão recebe dados sobre o sistema convidado do monitor de máquina virtual e os compara com os dados previamente armazenados.

Algumas medidas de tempo foram efetuadas sobre comandos básicos de usuário no sistema convidado, para permitir avaliar o impacto da arquitetura sobre o desempenho do sistema. O tempo de execução dos comandos `ps`, `find`, `ls` e `who` foram medidos em quatro situações distintas: a) no sistema hospedeiro, b) no sistema convidado original, c) no sistema convidado em modo aprendizagem e d) no sistema convidado em modo monitoração. Cada experimento foi repetido 10 vezes, com coeficientes de variação inferiores a 5% em todas as medidas.

A tabela 1 apresenta os tempos médios de execução para cada comando e seus custos relativos. O número de chamadas de sistema invocadas por cada processo também é apresentado. Observa-se que os tempos de execução no sistema convidado (linha b) são muito superiores aos observados no sistema hospedeiro (linha a); essa diferença é devida ao alto custo de virtualização apresentado pelo UML. Também são significativos os custos impostos pelas modificações no monitor de máquina virtual para interagir com os mecanismos externos de aprendizagem, monitoração e resposta (linhas c e d). Esse custo é devido à implementação realizada não estar otimizada.

**Tabela 1. Tempos médios de execução (em milissegundos)**

Comando		<code>ps -ef</code>	<code>find /&gt;/dev/null</code>	<code>ls -laR /&gt;/dev/null</code>	<code>who -B</code>
# chamadas de sistema		925	6958	18096	224
(a) sistema hospedeiro	tempo	22	41	166	2
	tempo	40	146	371	13
(b) sistema convidado	tempo	67	355	797	33
	custo relativo a (a)	82%	256%	123%	550%
(c) modo aprendizado	tempo	70	388	819	16
	custo relativo a (b)	67%	143%	115%	154%
(d) modo monitoração	tempo	75%	166%	121%	23%
	custo relativo a (b)				

Adicionalmente, para avaliar qualitativamente a efetividade da arquitetura em detectar e bloquear intrusões, alguns testes foram realizados usando *rootkits* populares na Internet (disponíveis em <http://www.antiserver.it/Backdoor-Rootkit/>).

Os *rootkits* modificam comandos do sistema operacional para impedir a detecção de intrusos (ocultando os processos, arquivos e conexões de rede do intruso) e para roubar informações digitadas como *logins* e senhas (através de modificações em comandos como `telnet`, `sshd` e `login`). Todas as ferramentas disponíveis em todos os *rootkits* indicados foram executadas com parâmetros *default* e foram detectadas em todas as execuções. Os testes evidenciaram a efetividade e complementaridade de ambos os mecanismos implementados pelo sistema: o mecanismo de detecção de intrusão detecta e impede a execução de programas conhecidos, mas adulterados, enquanto o controle de acesso impede a execução de programas desconhecidos, ou de programas lançados por usuários desconhecidos ou não-autorizados.

## 5. Trabalhos relacionados

A referência [Dunlap 2002] descreve uma experiência de uso de máquinas virtuais na segurança de sistemas. A proposta define uma camada intermediária entre o monitor e o sistema hospedeiro, denominada *Revirt*. Essa camada captura os dados enviados pelo processo *syslogd* (o *daemon* padrão Unix que gerencia mensagens de log) do sistema convidado e os envia ao sistema hospedeiro para armazenamento e futura análise. Entretanto, se o sistema convidado for invadido, o processo *syslogd* pode ser terminado ou subvertido e as mensagens de log podem ser manipuladas para ocultar o invasor, tornando o sistema sem efeito.

O trabalho que mais se aproxima de nossa proposta está descrito em [Garfinkel 2003]. Ele define uma arquitetura para a detecção de intrusão em máquinas virtuais denominada VMI-IDS (*Virtual Machine Introspection Intrusion Detection System*). Nossa proposta permite analisar processos isolados, detectando comportamentos anômalos e bloqueando intrusões providas de processos comprometidos. Dessa forma, perturbações nos demais processos convidados são minimizadas. Além disso, não é necessário suspender ou reiniciar a máquina virtual em caso de intrusão. Outra característica exclusiva de nossa proposta é o uso de um modelo de autorização (ACL) para usuários e processos construído automaticamente durante a fase de aprendizado do sistema.

## 6. Conclusão

O objetivo principal do projeto, restringir a execução de processos suspeitos na máquina virtual e conseqüentemente evitar o comprometimento do sistema, foi alcançado com o protótipo atual. Todavia, trabalhos complementares devem ser realizados para melhorar o desempenho dos mecanismos atuais de detecção e resposta, e diminuir o custo imposto às aplicações. No momento estamos estudando melhorias no código do UML e nos algoritmos implementados.

## Referências

- Allen J. et al. *State of the Practice of Intrusion Detection Technologies*. Technical Report CMU/SEI-99-TR028. Carnegie Mellon University, 1999.
- Blunden B. *Virtual Machine Design and Implementation in C/C++*. Wordware Publ. Plano, Texas – USA, 2002.
- Chen P., Noble B. *When Virtual is Better than Real*. Proceedings of the Workshop on Hot Topics in Operating Systems – HotOS, 2001.
- Dike J. *A User-mode port of the Linux Kernel*. Proceedings of the 4th Annual Linux Showcase & Conference. Atlanta – USA, 2000.
- Dunlap G. et al. *ReVirt: Enabling Intrusion Analysis through Virtual-Machine Logging and Replay*. Proceedings of the Symposium on Operating Systems Design and Implementation, 2002.
- Forrest S., Hofmeyr S., Somayaji A. *A sense of self for Unix processes*. Proceedings of the IEEE Symposium on Research in Security and Privacy, 1996.
- Garfinkel T., Rosenblum M. *A Virtual Machine Introspection Based Architecture for Intrusion Detection*. Proceedings of the Network and Distributed System Security Symposium, 2003.
- Goldberg R. *Architecture of Virtual Machines*. AFIPS National Computer Conference. New York – NY – USA, 1973.
- Hofmeyr S., Forrest S., Somayaji A. *Intrusion Detection using Sequences of System Calls*. Journal of Computer Security, 6:151–180, 1998.
- VMware Inc. *VMware Technical White Paper*. Palo Alto – CA – USA, 1999.