Anais IV WSEG – Workshop em Segurança de Sistemas Computacionais, SBRC – Gramado, RS – 2004. P. 63-74

Proteção de detectores de intrusão através de máquinas virtuais

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 conseqüência, 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. Detectores de intrusão baseados em rede analisam dados coletados da rede buscando detectar atividades maliciosas, podendo ser instalados em máquinas dedicadas e relativamente inacessíveis a intrusos. Por outro lado, sistemas de detecção de intrusão baseados em host analisam dados locais coletados em um sistema. Por executarem no próprio

sistema a monitorar, eles são particularmente vulneráveis a intrusos bem sucedidos, que podem desabilitá-los ou modificá-los para esconder sua presença e suas atividades.

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 freqüente 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).

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 detector de intrusão 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 detector de intrusão 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.

Este artigo está assim estruturado: a seção 2 relembra alguns conceitos de máquinas virtuais usados neste trabalho; a seção 3 define os mecanismos básicos de detecção de intrusão; a seção 4 detalha a proposta do trabalho; a seção 5 apresenta resultados da implementação do protótipo e a seção 6 apresenta e discute os trabalhos relacionados.

2. Máquinas Virtuais

Uma *máquina virtual* (*Virtual Machine – VM*) é definida em [Popek 1974] como sendo uma duplicata isolada e eficiente de uma máquina real. Um ambiente de máquina virtual é criado por um *monitor de máquinas virtuais* (*Virtual Machine Monitor – VMM*), também denominado um "sistema operacional para sistemas operacionais" [Kelem 1991]. O monitor cria uma ou mais máquinas virtuais sobre uma mesma máquina real. Cada máquina virtual provê as funcionalidades necessárias para um *sistema operacional convidado* (*guest operating system*) que acredita estar executando sobre uma plataforma convencional de

hardware[1]. Usos típicos de sistemas de máquinas virtuais incluem o desenvolvimento e teste de novos sistemas operacionais, a execução simultânea de vários sistemas distintos sobre o mesmo hardware e a consolidação de servidores [Sugerman 2001].

Existem duas arquiteturas clássicas para a construção de sistemas de máquinas virtuais: nos sistemas de tipo I, o monitor é implementado entre o hardware e os sistemas convidados; nos sistemas de tipo II o monitor é implementado como um processo de um sistema operacional convencional subjacente, denominado *sistema hospedeiro* (*host operating system*). Este artigo considera a aplicação de ambientes de máquinas virtuais de tipo II na segurança de sistemas.

Micro-processadores de mercado não provêem suporte adequado à virtualização do hardware. Conseqüentemente, o custo de virtualização pode ultrapassar 50% do tempo total de processamento [Blunden 2002, Dike 2000, VMWare 1999]. Entretanto, pesquisas recentes têm obtido reduções significativas nesses custos, levando-o a patamares inferiores a 10% [King 2002, King 2003, Whitaker 2003]. Um bom exemplo é o projeto *Xen* [Barham 2003], que obteve custos de virtualização inferiores a 3% para a execução de sistemas convidados *Linux*, *FreeBSD* e *Windows XP*. Esses trabalhos abrem muitas perspectivas quanto ao uso de ambientes de máquinas virtuais em ambientes de produção.

3. 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 com ameaças.

As seqüências de chamadas de sistema executadas pelos processos 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. Na proposta apresentada em [Forrest 1996, Hofmeyr 1998], as chamadas de sistema invocadas

---

[1] A JVM (*Java Virtual Machine*) não constitui um ambiente de máquinas virtuais no contexto deste trabalho; esse ambiente constitui em realidade o *emulador* de um processador abstrato de *bytecodes*.

por um processo são registradas seqüencialmente, descartando seus parâmetros. Essa "história de execução" é então transformada em um conjunto de seqüências de chamadas com comprimento *k*. O conjunto de todas as seqüências de comprimento *k* possíveis para um dado programa define o comportamento normal do mesmo. Toda seqüência de chamadas de sistema emitida por execuções subseqüentes daquele programa e não encontrada em seu comportamento normal (as seqüências previamente registradas) é considerada uma anomalia.

Para ilustrar essa técnica, consideremos um programa Unix que emitiu as seguintes chamadas de sistema durante sua execução:

```
[open read mmap mmap open read mmap]
```

Adotando por exemplo *k* = 3, o seguinte conjunto de seqüências é obtido:

```
[open read mmap] [read mmap mmap] [mmap mmap open]
[mmap open read]  [open read mmap]
```

Caso em uma execução posterior esse programa emita uma seqüência distinta, como `[open open read]`, ele deverá ser colocado sob suspeita. Apesar do conjunto de chamadas de sistema ser fortemente dependente do sistema operacional e do procedimento de captura do comportamento normal de um programa ser potencialmente trabalhoso, esse método apresenta uma eficiência satisfatória, como demonstrado por seus autores em [Hofmeyr 1998].

4. Protegendo detectores de intrusão

Como mostrado na seção precedente, detectores de intrusão baseados em host são vulneráveis a ataques locais, porque o intruso pode desativá-los ou modificá-los. O uso de máquinas virtuais pode prover uma solução satisfatória para esse problema. Esta seção apresenta uma proposta de arquitetura que 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 e assim protegê-lo de ataques.

A idéia central da proposta é encapsular o sistema a monitorar em uma máquina virtual. O detector de intrusão e o mecanismo de resposta são então implementados fora da máquina virtual, ou seja, fora do alcance de invasores. Esta proposta prevê o uso de um monitor de máquina virtual de tipo II, de forma que a detecção e a resposta sejam implementadas por processos do sistema operacional hospedeiro. A figura 1 ilustra os principais componentes da arquitetura proposta.

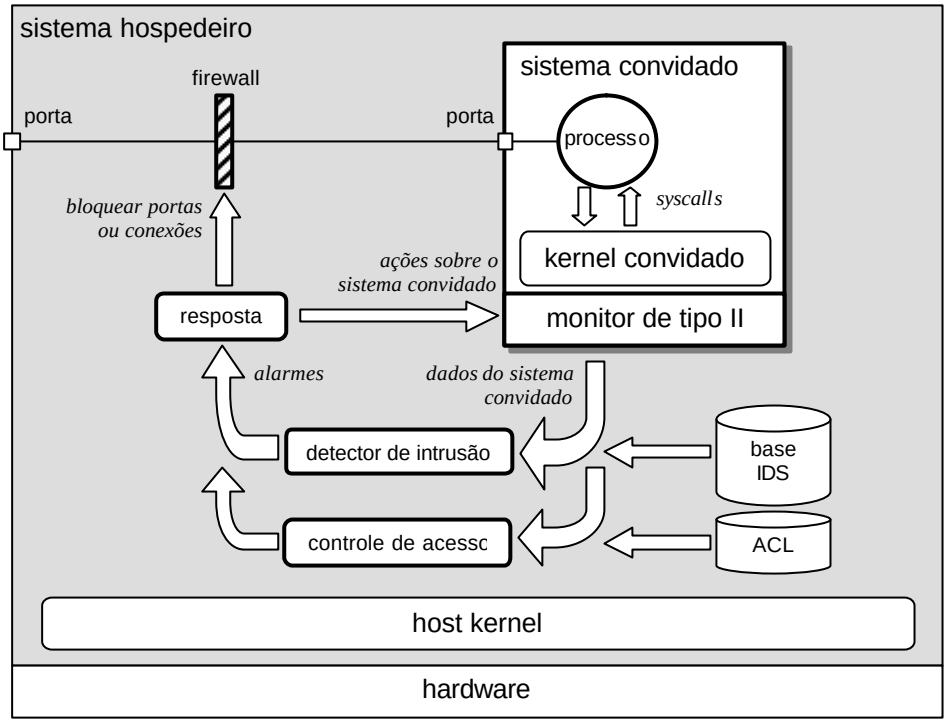

Fig. 1. Arquitetura proposta

Os principais componentes da arquitetura são: o *detector de intrusão*, que compara o dados coletados do sistema convidado com uma base de dados previamente construída, o módulo de *controle de acesso*, que verifica se os processos e usuários do sistema convidado existem em uma lista de controle de acesso (ACL) previamente estabelecida, e o módulo de *resposta*, que recebe alarmes gerados pelos módulos anteriores e os transforma em ações sobre o sistema convidado e/ou o *firewall* do sistema hospedeiro.

A interação dos processos do sistema convidado com o mundo externo é efetuada exclusivamente via rede, através de um *firewall* de software gerenciado pelo núcleo do sistema operacional hospedeiro (como por exemplo o *iptables* do sistema Linux). Sob a ótica do sistema convidado, trata-se de um *firewall* externo, portanto inacessível a intrusos.

A interação entre o sistema convidado e o sistema de detecção e resposta é realizada através do monitor de máquina virtual. Dois tipos de interação são definidos: *monitoração*, na qual dados do sistema convidado são fornecidos pelo monitor para os sistemas externos de detecção de intrusão e controle de acesso, e *resposta*, permitindo ao módulo de resposta agir sobre o sistema convidado em reação a intrusões. Além de ações sobre o sistema convidado, o módulo de resposta pode também interagir sobre o *firewall* externo, bloqueando portas e conexões conforme necessário.

Deve-se observar que, apesar do protótipo atual implementar a abordagem de detecção de anomalia, usando o algoritmo de detecção apresentado na seção anterior, a arquitetura apresentada na figura 1 é suficientemente genérica para ser usada com outros algoritmos de detecção de intrusão por anomalia e também por assinatura.

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 processo

executando no sistema convidado e seus respectivos usuários são registrados como usuários e processos autorizados, gerando assim uma lista de controle de acesso (ACL – *Access-Control List*). O sistema também registra as seqüências de chamadas de sistema dos processos convidados em uma base de dados. O modo aprendizado permite, portanto, registrar o "comportamento normal" do sistema, coletando e organizando a informação necessária para a detecção de intrusão.

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. O protótipo atual analisa as seqüências de chamadas de sistema emitidas pelos processos convidados, usando o algoritmo apresentado em [Hofmeyr 1998]. Se uma seqüência emitida por um processo não for encontrada na base de seqüências conhecidas para seu respectivo programa, uma anomalia é relatada e aquele processo é declarado suspeito. Além disso, processos e usuários em desacordo com a ACL previamente gerada são declarados suspeitos pelo módulo de controle de acesso.

Processos suspeitos sofrem restrições em seu acesso aos recursos do sistema convidado, para prevenir ações perigosas e bloquear eventuais intrusões. Essas restrições são implementadas negando o acesso do processo suspeito a determinadas chamadas de sistema. Os trabalhos [Bernaschi 2000, Bernaschi 2002] classificam as chamadas de sistema do UNIX em grupos de funcionalidades (por exemplo: comunicação, sistema de arquivos, gerência de memória) e níveis de ameaça assim definidos:

- *Nível 1*: chamadas de sistema que podem ser usadas para obter acesso privilegiado ao sistema operacional;
- *Nível 2*: chamadas de sistema que podem ser usadas para ataques de negação de serviço;
- *Nível 3*: chamadas capazes de corromper processos específicos;
- *Nível 4*: não representam perigo para a segurança do sistema.

Essa classificação está sendo usada neste trabalho da seguinte forma: todas as chamadas de sistema classificadas no nível 1 são sistematicamente negadas aos processos declarados suspeitos. Essas chamadas estão definidas na tabela 1. Esse mecanismo foi implementado modificando o monitor de máquina virtual, que pode interceptar as chamadas de sistema invocadas pelos processos convidados. Usando essa abordagem, um processo suspeito pode ser isolado sem causar impacto em outros processos do sistema convidado que não dependam dele.

Tabela 1. Chamadas de sistema negadas a processos suspeitos

Grupo	**Chamadas de sistema**
Sistema de arquivos e dispositivos	`open link unlink chmod lchown rename fchown chown mknod mount symlink fchmod`
Gerência de processos	`execve setgid setreuid setregid setgroups setfsuid setfsgid setresuid setresgid setuid`
Gerência de módulos	`init_module`

A arquitetura aqui apresentada mantém o sistema de detecção e resposta fora do alcance de intrusos. Entretanto, para garantir a segurança do sistema, é importante observar que as interações com o sistema convidado devem ser realizadas sempre através do monitor de máquina virtual. Além disso, o monitor deve estar inacessível aos processos convidados (de acordo com [Kelem 1991], esta é uma propriedade conceitual dos monitores de máquinas virtuais). Finalmente, todos os serviços de rede devem ser providos pelos processos convidados; o acesso via rede ao sistema hospedeiro deve ser cuidadosamente controlado.

5. 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]. Deve-se observar que o desempenho do UML não é comparável ao de produtos comerciais como o VMWare [VMWare 1999], mas seu código-fonte é aberto e tornou possível esta implementação. O código do 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 usado o *Suse Linux Professional* 9.0 (kernel 2.4.21), e como sistema convidado foi usado um *Linux Debian* 2.2 com kernel 2.6.1.

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 quatros 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 2 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 2. Tempos médios de execução (em milissegundos)

Comando		`ps –ef`	`find / >/dev/null`	`ls -laR / >/dev/null`	`who –B`
# chamadas de sistema		925	6958	18096	224
(a) sistema hospedeiro	tempo	22	41	166	2
(b) sistema convidado	tempo	40	146	371	13
	custo relativo a (a)	82%	256%	123%	550%

(c) modo aprendizado	tempo	67	355	797	33
	custo relativo a (b)	67%	143%	115%	154%
(d) modo monitoração	tempo	70	388	819	16
	custo relativo a (b)	75%	166%	121%	23%

Adicionalmente, para avaliar qualitativamente a efetividade da arquitetura em detectar e bloquear intrusões, alguns testes foram realizados usando *rootkits* populares na Internet (descritos na tabela 3 e disponíveis em *http://www.antiserver.it/Backdoor-Rootkit/*).

Tabela 3. Rootkits usados para testar o protótipo

Nome	**Descrição**
`FK 0.4`	Linux kernel module rootkit e trojan SSH.
`Adore`	Oculta arquivos, diretórios, processos, fluxos de rede. Instala um *backdoor* e um programa de usuário para controlar o sistema.
`ARK 1.0`	*Ambient's Rootkit for Linux*. Inclui versões com *backdoor* dos comandos syslogd, login, sshd, ls, du, ps, pstree, killall, e netstat.
`Knark v.2.4.3`	Oculta arquivos, fluxos de rede, processos e redireciona a execução de programas.
`hhp-trosniff`	Conjunto completo de modificações do ssh, ssh2m sshd2 e openssh, para extrair e registrar origem, destino, nome do *host*, nome do usuário e senha.
`login`	*Universal login trojan* – Utilizado para registrar nomes e senhas de acesso em Linux.

Esses 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.

6. Trabalhos relacionados

O artigo [Chen 2001] enumera alguns benefícios que o uso de máquinas virtuais pode trazer para a segurança e compatibilidade de sistemas, como a captura e processamento de mensagens de log, a detecção de intrusão através do controle do estado interno da máquina virtual, ou facilidades para a migração de sistemas. Entretanto, o artigo não demonstra como essas propostas podem ser estruturadas e implementadas, nem analisa seu impacto no desempenho do sistema.

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*). Sua abordagem considera o uso de um monitor de tipo I, executando diretamente sobre o hardware. O detector de intrusão executa em uma máquina virtual privilegiada e analisa dados extraídos das demais máquinas virtuais, buscando evidências de intrusão. Somente o estado interno de baixo nível de cada máquina virtual é analisado, sem levar em conta as atividades desenvolvidas pelos processos convidados. Além disso, A capacidade de resposta do sistema é limitada: caso exista suspeita de intrusão, a máquina virtual suspeita é suspensa para análise detalhada; se a intrusão for confirmada, a máquina virtual é reiniciada a partir de um estado anterior seguro previamente salvo.

Essa abordagem difere de nossa proposta em vários aspectos, como a natureza e granularidade dos dados coletados, os métodos de detecção de intrusão, o mecanismo de controle de acesso e a capacidade de resposta mais específica. Nossa proposta permite analisar processos isolados, detectando comportamentos anômalos e bloqueando intrusões provindas 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.

Uma abordagem alternativa para o isolamento de detectores de intrusão seria o uso de núcleos com capacidade de definir contextos de usuário isolados entre si (núcleos com essa capacidade são descritos em [Pfitzmann et al 2001; Embry 2001; Linux VServer Project 2004]). Neste caso, o detector de intrusão seria instalado em um contexto privilegiado, com visão e possibilidade de ação sobre os demais contextos. Essa abordagem pode ter bons resultados em termos de desempenho, embora imponha o mesmo sistema operacional a todos os contextos de aplicação, o que reduz sua flexibilidade.

7. Conclusão

Este artigo descreve uma proposta para aumentar a segurança de sistemas computacionais usando máquinas virtuais. A base da proposta é monitorar as ações dos os processos convidados através de um mecanismo de detecção de intrusão externo à máquina virtual. Os dados utilizados na detecção de intrusão são obtidos do monitor de máquina virtual e analisados por um sistema de detecção de intrusão no sistema hospedeiro. O detector de intrusão é inacessível aos processos convidados e não pode ser comprometido por intrusos. O detector de intrusão pode monitorar a atividade de processos isolados, enquanto o módulo

de resposta pode restringir a atividade de processos suspeitos sem perturbar outros processos.

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.

Outro aspecto a ser considerado é a definição de formas mais flexíveis de interação com o núcleo convidado, que permitam, por exemplo, terminar, suspender ou modificar as prioridades de processos suspeitos. As interações entre o módulo de resposta e o *firewall* do sistema hospedeiro também precisam ser detalhadas e implementadas.

Para facilitar o uso do sistema, o próximo protótipo irá permitir que os modos de monitoração e aprendizagem ocorram simultaneamente, para processos distintos. Isso irá permitir ao sistema "aprender" o comportamento de uma nova aplicação enquanto monitora os demais processos. Outras questões a estudar incluem a implementação de mecanismos de detecção de intrusão baseados em outros dados relevantes, como o tráfego de rede gerado pelos processos convidados.

Referências

Allen J., Christie A., Fithen W., McHugh J., Pickel J., Stoner E. *State of the Practice of Intrusion Detection Technologies*. Technical Report CMU/SEI-99-TR028. Carnegie Mellon University, 1999.

Barham P., Dragovic B., Fraser K., Hand S., Harris T., Ho A., Neugebauer R., Pratt I., Warfield A. *Xen and the Art of Virtualization*. Proceedings of the ACM Symposium on Operating Systems Principles – SOSP, 2003.

Bernaschi M., Grabrielli E., Mancini L. *Operating System Enhancements to Prevent the Misuse of System Calls*. Proceedings of the ACM Conference on Computer and Communications Security, 2000.

Bernaschi M., Grabrielli E., Mancini L. *REMUS: A Security-Enhanced Operating System*. ACM Transactions on Information and System Security. Vol 5, number 1, 2002.

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., King S., Cinar S., Basrai M., Chen P. *ReVirt: Enabling Intrusion Analysis through Virtual-Machine Logging and Replay*. Proceedings of the Symposium on Operating Systems Design and Implementation – OSDI, 2002.

Embry R. *FreeVSD Enables Safe Experimentation*. Linux Journal, Issue 87, July 2001.

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 – NDSS, 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.

Kelem N., Feiertag R. *A Separation Model for Virtual Machine Monitors*. Proceedings of the IEEE Computer Society Symposium on Research in Security and Privacy, pages 78-86, 1991.

King S., Chen P. *Operating System Extensions to Support Host Based Virtual Machines*. Technical Report CSE-TR-465-02, University of Michigan, 2002.

King S., Dunlap G., Chen P. *Operating System Support for Virtual Machines*, Proceedings of the USENIX Annual Technical Conference, 2003.

Linux VServer Project. http://www.linux-vserver.org, 2004.

Pfitzmann B., Riordan J., Stüble C., Waidner M., Weber A. *The PERSEUS System Architecture*. Research Report RZ 3335 04/09/01, IBM Research Division, Zurich, April 2001.

Popek G., Goldberg R. *Formal Requirements for Virtualizable Third Generation Architectures*. Communications of the ACM. Volume 17, number 7, pages 412-421, 1974.

Sugerman J., Ganesh V., Beng-Hong L. *Virtualizing I/O Devices on Vmware Workstation's Hosted Virtual Machine Monitor*. Proceedings of the USENIX Annual Technical Conference, 2001.

VMware Inc. *VMware Technical White Paper*. Palo Alto – CA – USA, 1999.

Whitaker A., Shaw M., Gribble S. *Denali: A Scalable Isolation Kernel*. Proceedings of the 10th ACM SIGOPS European Workshop, Saint-Emilion – France, 2002.