TuxGuardian: Um firewall de host voltado para o usuário final

Bruno Castro da Silva, Raul Fernando Weber

Instituto de Informática – Universidade Federal do Rio Grande do Sul
Caixa Postal 15064 – 90501-970 Porto Alegre, RS

{bcs,weber}@inf.ufrgs.br

Abstract. *This paper describes the implementation of TuxGuardian, a host firewall developed after the observation that Linux security applications were not tailored for lay users. By using the security mechanisms adopted in the Linux kernel 2.6, TuxGuardian is able to implement access control policies to the network resources. This way, it is capable of identifying and controlling every application that tries to access the network, forbidding transmission and reception of data by unauthorized applications. This firewall has been developed in a three-layered architecture of independent modules, and is able to operate with or without user intervention. Also, TuxGuardian's modular design eases the task of adding new features and functionality.*

Resumo. *Este artigo descreve a implementação do TuxGuardian, um firewall de host desenvolvido a fim de possibilitar a gerência de segurança por parte de usuários leigos. Através da utilização de mecanismos de segurança presentes no kernel 2.6 do Linux, o TuxGuardian possibilita a implementação de políticas de controle de acesso aos recursos de rede. Desta maneira, é capaz de identificar e gerenciar aplicações que tentam acessar a rede, impedindo que aplicativos maliciosos possam gerar ou receber tráfego pela Internet. O firewall foi desenvolvido na forma de módulos independentes, em uma arquitetura de três camadas, e é capaz de operar com ou sem o auxílio do usuário. O* design *modular do TuxGuardian, licenciado pela GNU/GPL, também permite a fácil expansão de suas funcionalidades.*

1. Introdução

A crescente popularização da Internet e dos serviços disponibilizados através da rede fez com que a busca por sistemas seguros se tornasse uma necessidade indiscutível. Prova disso é a ampla difusão de mecanismos de controle de tráfego, tais como firewalls e sistemas de detecção de intrusão e, mais recente, a utilização de sistemas conhecidos como "Firewalls de Host". Estes nada mais são do que mecanismos de fácil operação desenvolvidos com intuito de proteger individualmente cada computador. Em oposição aos dispositivos convencionais de segurança, como aqueles baseados na filtragem de pacotes, os firewalls de host tornam o computador seguro através do controle de quais aplicações têm autorização para receber e transmitir dados pela rede.

Através de um estudo sobre os aplicativos de segurança existentes para Linux, constatou-se que não existem soluções que atendam às expectativas do usuário final. A

maior parte dos aplicativos existentes implementam interfaces gráficas para programas já conhecidos, como o Netfilter, *nmap* e *tcpdump*. Tendo em vista que todos os programas estudados exigem do usuário uma série de conhecimentos acerca do modo como as redes operam, pode-se concluir que nenhum deles é realmente voltado para o usuário leigo.

Com base nessa constatação, foi desenvolvido um firewall de host extensível e de fácil utilização, chamado TuxGuardian. O TuxGuardian foi desenvolvido para plataforma Linux e é software livre.

Nas seções seguintes será apresentado o modo de funcionamento da ferramenta desenvolvida. Primeiramente será exposta a arquitetura em três camadas utilizada no desenvolvimento do firewall, assim como os aplicativos que implementam cada camada do TuxGuardian. Serão discutidos os motivos e vantagens de se adotar certas decisões de projeto, assim como o funcionamento do protocolo utilizado para permitir a comunicação entre as partes que compõem o TuxGuardian. Por fim, serão listadas algumas situações nas quais o uso deste firewall de host é vantajoso, assim como expectativas de trabalhos futuros e conclusões.

2. Descrição da ferramenta implementada

2.1. Arquitetura de três camadas

O TuxGuardian foi desenvolvido de maneira modular, o que significa que existem várias componentes que cooperam para que a funcionalidade do firewall seja obtida. Foram especificadas três camadas distintas, cada uma com uma tarefa bem determinada, assim como um protocolo de comunicação utilizado na troca de informações entre essas camadas.

O *módulo de segurança* é a camada do TuxGuardian que opera em mais baixo nível, fornecendo ao firewall a capacidade de monitorar aplicações que tentam acessar a rede, tanto como clientes quanto como servidoras. A fim de tornar a implementação do módulo de segurança o menos invasiva possível, optou-se por manter todas informações de segurança fora do *kernelspace*, o que justifica a existência da segunda camada na arquitetura do TuxGuardian.

Essa segunda camada é implementada na forma de um *daemon*, e é responsável por armazenar as informações a respeito de quais aplicativos são considerados confiáveis para acessar a rede. O daemon é um programa de usuário que é contatado pelo módulo de segurança sempre que alguma decisão crítica de segurança precisa ser tomada. Caso o daemon não possua informações suficientes para responder de forma definitiva ao módulo, a consulta pode ser repassada para um *frontend*.

O frontend é a terceira camada do TuxGuardian, e seu objetivo principal é fornecer os meios necessários para que possa haver interação em tempo real com o usuário. O frontend é uma peça opcional para o funcionamento do firewall de host, e seu uso é determinado pelas necessidades do usuário ou do administrador.

2.2. O módulo de segurança

O módulo de segurança nada mais é do que um módulo carregável convencional do Linux. A principal função do módulo de segurança é detectar e tratar certos eventos de rede

considerados importantes, tais como uma aplicação tentando atuar como servidora. Note-se que o TuxGuardian não analisa diretamente o tráfego de rede, e sim as operações que ocorrem no nível do aplicativo. Por uma decisão de projeto, as operações monitoradas são aquelas que envolvem a criação e utilização de sockets. Isto permite que operações críticas, como um *connect* ou *listen*, sejam descobertas antes de mesmo serem realmente executadas. Desta forma, impede-se a geração de tráfego indesejado, eliminando a necessidade de mecanismos que posteriormente analisem o tráfego e removam os pacotes suspeitos.

O monitoramente destas operações é possível devido à utilização do *framework* LSM (Linux Security Modules). O LSM consiste em uma ferramenta recentemente adotada no kernel *mainstream* do Linux, e que foi desenvolvida a fim de facilitar a criação de políticas de controle de acesso. A grande vantagem de se utilizar o LSM, ao invés de se optar por opções invasivas, como a recompilação do kernel ou a sobreposição de entradas na tabela de chamadas de sistema, é que a mediação aos objetos internos do kernel se dá de maneira organizada. O LSM insere várias *callbacks* em pontos estratégicos do código do kernel, de maneira que funções podem ser chamadas no momento em que ocorrerem acessos a determinados objetos internos do kernel. Por objeto interno do kernel entende-se estruturas como *inodes*, sockets, *tasks*, etc. No caso do TuxGuardian, foram desenvolvidas *callbacks* (também chamadas de *hooks*) que são disparadas quando ocorrem operações sobre sockets. Assim, é possível delegar a um módulo carregável a tarefa de decidir se determinada operação deve ser permitida ou não, de acordo com a política de segurança definida pelo usuário.

Os dois *hooks* definidos no módulo de segurança do TuxGuardian são os seguintes[1]: **socket_create** e **socket_listen**. Ambos são utilizados a fim de mediar operações sobre sockets da família de protocolos PF_INET e PF_INET6, de modo que todo o tráfego IPv4 e IPv6 pode ser monitorado. Comunicações locais via sockets não são verificadas.

O hook *socket_create* define uma função responsável por tomar decisões de segurança sempre que um novo socket for criado. Esta função dá ao TuxGuardian condições de impedir que aplicativos não autorizados possam criar meios que lhe permitam o acesso à rede, evitando que estes aplicativos transmitam e recebam dados pela Internet.

O segundo *hook* (*socket_listen*) tem funcionalidade semelhante, mas é acionado sempre que uma aplicação tenta atuar como servidora através da chamada *listen*, a qual é utilizada para inicializar uma fila de espera de conexões.

Após detectar um evento de rede através das *callbacks* citadas anteriormente, o módulo de segurança utiliza rotinas auxiliares que transmitem informações de contexto para o daemon. Com base nestas informações, é possível determinar qual é a aplicação que está tentando acessar a rede, e também para que tarefas esta aplicação é confiável.

2.3. O daemon

O daemon corresponde à camada do TuxGuardian responsável por manter atualizadas todas as informações de segurança do firewall de host. Para isso, um arquivo de configuração é lido e seus dados são transferidos para uma tabela hash mantida em memória. Esta

[1]na verdade mais *hooks* foram definidos, a fim de manter o suporte à lógica de *capabilities* (padrão POSIX.1e), utilizada por programas como o *named* e o *sendmail*.

tabela hash também é responsável por armazenar permissões de segurança transitórias, ou seja, permissões de acesso que são válidas somente durante a execução atual do firewall.

O daemon consiste em uma aplicação servidora capaz de receber conexões do módulo de segurança. Essas conexões são estabelecidas a fim de que o módulo de segurança possa fazer consultas (*queries*) a respeito de quais permissões de acesso à rede uma determinada aplicação possui.

No caso de comunicações módulo-daemon, as *queries* carregam na sua área de dados o identificador do processo (PID) que gerou a tentativa de acesso à rede. A partir deste PID, o daemon consulta a partição virtual /proc e determina qual é o binário que gerou o processo em questão. Com base nesta informação, é possível também calcular o hash MD5 da aplicação, de maneira que a integridade dos aplicativos pode ser garantida. A grande vantagem de se manter o hash MD5 de cada aplicação é que essa informação permite com que o TuxGuardian proíba programas adulterados, propositalmente ou não, de acessarem a Internet.

O arquivo de configuração do TuxGuardian é formado por entradas com o seguinte formato:

- path do aplicativo
- hash MD5 do binário
- permissões de segurança

Até o momento, as permissões disponíveis são: **PERMIT_APP**, **DENY_APP**, **PERMIT_SERVER** e **DENY_SERVER**.

As duas primeiras dizem respeito às permissões que o aplicativo têm para gerar e receber tráfego de rede. As duas últimas opções especificam permissões que um aplicativo têm para atuar como servidor. As opções PERMIT_APP e DENY_APP, quando utilizadas sozinhas, definem as permissões que um aplicativo tem enquanto cliente. Note-se que a opção DENY_APP impede completamente o acesso do aplicativo à rede, tanto como cliente quanto como servidor.

2.4. Protocolo de comunicação intercamadas

A troca de dados que ocorre entre as camadas do TuxGuardian segue o padrão definido pelo protocolo TGP (TuxGuardian Protocol). Este protocolo define mensagens de 10 bytes formadas por 4 campos:

- sender
- seqno
- query_type
- query_data

O campo *sender* especifica qual a camada do TuxGuardian originou determinada mensagem. Existem três opções de valores, correspondentes aos três aplicativos que implementam as funcionalidades do TuxGuardian: TG_MODULE, TG_DAEMON e TG_FRONTEND. Dessa maneira, cada camada, ao enviar um bloco de dados para outra, identifica-se como sendo a remetente.

O campo *seqno*, por sua vez, armazena um número de seqüência único para o bloco de dados, sendo utilizado caso haja a necessidade de identificar unicamente as *queries* recebidas.

O terceiro campo, *query_type*, define qual é a natureza da *query* encapsulada no bloco de dados. As opções definidas até o presente momento são as seguintes:

1. TG_ASK_PERMIT_APP: identifica uma *query* enviada pelo módulo ao daemon solicitando informações sobre a confiabilidade de determinada aplicação;
2. TG_RESPOND_PERMIT_APP: identifica uma resposta à *query* citada acima, sendo que a resposta propriamente dita é armazenada no campo *query_data*;
3. TG_PERMIT_SERVER: identifica uma *query* utilizada para determinar se uma aplicação pode ou não atuar como servidora na máquina (isto é, se essa aplicação possui permissão para receber conexões remotas);
4. TG_RESPOND_PERMIT_SERVER: identifica uma resposta à *query* citada acima;

Note que os três campos citados até agora servem apenas para caracterizar uma mensagem. Os dados propriamente ditos são armazenados no campo *query_data*, cuja semântica é definida em função do remetente da *query* e do conteúdo do campo *query_type*. O campo *query_data* pode conter um PID (para os casos em que se solicita ao daemon informações a respeito da confiabilidade de determinada aplicação), e também os valores YES e NO. A semântica destes valores pode ser melhor especifica através de sufixos, como nos exemplos a seguir[2]:

- *NO_WRONG_HASH*: informa ao módulo que determinada aplicação não deve ter acesso à rede pois sua integridade foi comprometida;
- *NO_USER_FORBID*: informa ao daemon que o usuário, quando consultado a respeito de determinada aplicação, determinou que ela não deve ter acesso à rede;
- *YES_SAVE_IN_FILE*: informa ao daemon que a resposta obtida após a interação com o usuário deve ser gravada nos arquivos de configuração, e que o acesso requisitado pela aplicação deve ser concedido;
- *NO_ACCESS_IS_DENIED*: informa ao módulo que o daemon foi explicitamente configurado para negar o acesso à determinada aplicação.

Um exemplo de comunicação entre as camadas é apresentado na figura 1.

É válido ressaltar que o TuxGuardian assume uma política restritiva de segurança: os acessos são negados sempre que não houver informações suficientes para que a tomada de decisão seja feita. Isso pode ocorrer, por exemplo, quando o frontend não estiver disponível, ou quando ocorrer um erro inesperado no daemon. É interessante ressaltar também que, por uma razão de segurança, todos os acessos negados são registrados nos logs do sistema, tipicamente em /var/log/messages, sob nível KERN_INFO.

2.5. O frontend

A última parte que compõe o TuxGuardian, e que na verdade é opcional para o seu funcionamento, é o frontend. O frontend nada mais é do que um aplicativo utilizado para obter a opinião do usuário nos casos em que o daemon não é capaz de agir de forma autônoma.

A implementação de um frontend para o TuxGuardian é bastante simples, já que sua única tarefa é fornecer uma interface capaz de traduzir *queries* TGP para uma forma

[2]a lista apresentada não é exaustiva. Para uma relação completa dos valores possíveis, consulte [Silva, 2004]

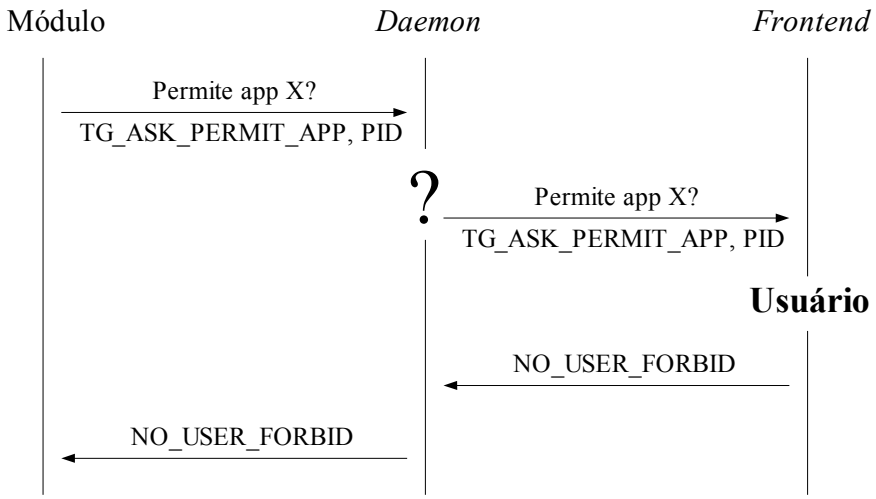

Figura 1: Exemplo de interação entre as 3 camadas do TuxGuardian.

que o usuário final seja capaz de entender. Após obter a opinião do usuário sobre alguma aplicação, o frontend deve repassar a resposta para daemon, já que este é o responsável por efetivamente armazenar as políticas de segurança. No caso do frontend não estar disponível, o TuxGuardian opera apenas com as regras definidas estaticamente no seu arquivo de configuração.

Um exemplo da interação do frontend com o usuário pode ser visto na figura 2. Consultas como a apresentada na figura 2 são exibidas no ambiente gráfico do usuário sempre que o daemon solicitar intervenção humana para análise de uma aplicação suspeita.

Figura 2: O frontend solicitando intervenção do usuário.

A seguir pode-se ver alguns exemplos do tipo de saída gerada pelo daemon. No primeiro caso, em que a aplicação ping tenta atuar como cliente, o daemon não possui informações suficientes para tomar uma decisão, e então solicita auxílio do usuário através do frontend. Note que a resposta dada pelo usuário é armazenada nos arquivos de configuração do TuxGuardian. No segundo caso a aplicação sshd tenta atuar como servidora, mas o daemon está configurado estaticamente para impedir que isto ocorra. O daemon, em ambas a situações, repassa a decisão definitiva ao módulo de segurança.

```
Thu Jan 16 11:20:13 2004
    Query from module wants to know if
       PERMIT_APP /bin/ping
        with hash e9cb6b3fc38d7d7973fe641922366bd7
    Checking the permissions for the application..
    Daemon does not know if PERMIT_APP is ok...
    Forwarding query to frontend.. done!
      The config file has been updated!
    Got answer. User said YES
        Answer has been stored in hashtable (5).
    Sending YES to the module..

Sat Jan 16 11:21:26 2004
    Query from module wants to know if
       PERMIT_SERVER /usr/sbin/sshd
        with hash a93bd2d4293989531f411914c4e80707
    Checking the permissions for the application..
    Daemon configured to DENY this SERVER
    Sending NO to the module..
```

3. Exemplos de uso

São descritos a seguir, em uma lista não exaustiva, algumas situações nas quais o uso do TuxGuardian é aconselhado:

- proteção da máquina do usuário final nos casos em que este não deseja se proteger de nenhuma ameaça específica, mas é prudente o suficiente para não deixar que qualquer aplicativo acesse a rede de modo indiscriminado;
- proteção adicional em máquinas onde já existem mecanismos de segurança de mais baixo nível, como um filtro de pacotes, e o usuário deseja controlar as operações de rede efetuadas pelos aplicativos;
- identificação e isolamento de aplicações mal-intencionadas no exato momento em que elas tentam agir (ex: um *spyware* tentando transmitir dados confidenciais pela internet);
- proteção contra alterações intencionais em aplicativos que acessam a rede, garantindo a integridade e confiabilidade de todos programas utilizados para transmissão de dados pela internet;
- implementação da segurança interna dentro de uma rede, visto que os firewalls de filtro de pacotes quase sempre lidam com ameaças externas;
- criação de uma camada extra de proteção na rede, adicionalmente àquela geralmente provida pelo firewall da rede ou *gateway*. Essa proteção seria inerentemente distribuída, já que atuaria individualmente em cada host, o que resultaria em um impacto direto sobre a carga imposta ao firewall institucional.

Tendo em vista que o desenvolvimento do TuxGuardian ainda não está concluído, são listadas a seguir algumas tarefas pendentes que devem ser desenvolvidas em um futuro próximo:

- testes de performance e overhead sobre o sistema;
- desenvolvimento de métodos que garantam a integridade do módulo de segurança, assim como dos sockets locais que implementam comunicação intercamadas e do arquivo de configuração do daemon;
- desenvolvimento de rotinas que permitam a análise de integridade de aplicativos *linkados* dinamicamente;
- adição de controles mais detalhados sobre o tipo de tráfego e as portas envolvidas nas operações de rede monitoradas;
- identificação e tratamento de recursos de rede alocados antes da inicialização do TuxGuardian.

4. Conclusões

A recente popularização e crescimento dos sistemas interligados em rede fez com que diversos mecanismos de segurança tivessem que ser concebidos. Essa tendência é reforçada pela demanda por mecanismos que possam ser configurados por usuários leigos e que atuem em nível de host. A gerência de aplicações confiáveis proporciona uma boa solução para este problema, na medida em que constitui um mecanismo que não exige do usuário amplo conhecimento acerca do modo como as redes operam. O TuxGuardian é um firewall de host desenvolvido levando-se em conta exatamente essa filosofia, e que é efetivamente capaz de fornecer uma camada extra de segurança, em adição àquela provida por mecanismos clássicos de segurança.

Devido à falta de ferramentas deste tipo para plataforma Linux, o TuxGuardian dá o primeiro passo no sentido de disponibilizar mecanismos de segurança fáceis de usar e que visem exclusivamente o usuário final. A distribuição através da licença GNU/GPL, o desenvolvimento modular e a utilização de *frameworks* genéricos de segurança contribuem para que o TuxGuardian possa ser aperfeiçoado e adaptado para fins diversos.

Por fim, a utilização de um protocolo extensível de comunicação entre as camadas do TuxGuardian faz com que o firewall desenvolvido, embora tenha objetivos bastante específicos, constitua uma ferramenta flexível e de grande valia, tanto para o meio acadêmico quanto para a comunidade de usuários Linux em geral.

Referências

Biondi, P. (2002). Security at kernel level. In *Proceedings of the Free and Open source Software Developers' European Meeting*, Bruxelas, Bélgica.

Corbet, J. M. (2003). Porting drivers to the 2.5 kernel. In *Proceedings of the Linux Symposium*, Ottawa, Canada.

Silva, B. C. (2004). Estudo e desenvolvimento de um firewall de host. Trabalho de Conclusão de Curso, Universidade Federal do Rio Grande do Sul.

Wright, C. (2002). Linux security module framework. In *Proceedings of the Linux Symposium*, Ottawa, Canada.

Wright, C. and Cowan, C. (2003). Linux security modules: General security support for the linux kernel. In *11th USENIX Security Symposium*, San Francisco, USA.