MECANISMOS PARA CONTENÇÃO DE TRÁFEGO MALICIOSO DE SAÍDA EM HONEYNETS

Klaus Steding-Jessen
NIC BR Security Office – NBSO
Comitê Gestor da Internet no Brasil
jessen@nic.br

Cristine Hoepers
NIC BR Security Office – NBSO
Comitê Gestor da Internet no Brasil
cristine@nic.br

Antonio Montes
Lab. Associado de Computação e Matemática Aplicada
Instituto Nacional de Pesquisas Espaciais
montes@lac.inpe.br

RESUMO

Para impedir que uma honeynet seja utilizada por invasores para desferir ataques contra outras redes é essencial que mecanismos de contenção sejam implantados. Neste artigo são discutidas técnicas para contenção do tráfego de saída gerado por atividades maliciosas e é apresentada a implementação de uma ferramenta que emprega algumas destas técnicas.

ABSTRACT

In order to prevent a honeynet from being used by intruders to attack other networks, the use of contention mechanisms is essential. In this paper we discuss some techniques to contain malicious outbound traffic and present a tool developed that applies some of these techniques.

1 INTRODUÇÃO

Honeynets são ferramentas utilizadas para acompanhar ataques e atividades de invasores [1,2,3]. Uma vez que um invasor comprometa um *host* de uma *honeynet*, ele provavelmente tentará usá-lo para atacar outros sistemas conectados à Internet.

Para impedir que a *honeynet* seja utilizada para desferir ataques é essencial que mecanismos de contenção sejam implantados para barrar completamente o tráfego malicioso de saída ou diminuir a ameaça que ele possa representar para outras redes [4].

Neste artigo são discutidas atividades maliciosas que podem ser realizadas por invasores, técnicas para contenção do tráfego gerado por estas atividades e é apresentada a implementação de uma ferramenta que emprega algumas destas técnicas.

Este artigo está organizado como segue. Na seção 2 são descritas algumas técnicas de contenção de tráfego malicioso de saída. Na seção 3 é apresentada uma ferramenta que implementa algumas das técnicas apresentadas. Em seguida, na seção 4 são discutidas conclusões e trabalhos futuros.

2 TÉCNICAS DE CONTENÇÃO

Esta seção descreve algumas atividades maliciosas que podem ser empregadas por invasores e possíveis técnicas para conter o tráfego gerado por estas atividades.

2.1 Contenção de ARP Spoofing

O protocolo ARP (*Address Resolution Protocol*) é usado para mapear dinamicamente endereços IP em endereços usados pela camada de enlace, como por exemplo endereços *ethernet* [5].

Esses mapeamentos são mantidos em *cache* pelos *hosts* da rede, incluindo equipamentos como *bridges* e *switches*. Um invasor que possa emitir consultas ou respostas ARP forjadas, injetando informações falsas nos *caches* dos demais *hosts* de sua rede *ethernet*, é capaz de:

- receber o tráfego de rede destinado a outro *host*, repassando-o para o seu destino original em seguida (*Man-in-the-Middle*). Este tipo de ataque pode ser usado inclusive para atacar protocolos com criptografia, como o SSH;
- inserir novos pacotes no tráfego redirecionado, como por exemplo comandos em sessões interativas;
- descartar pacotes, como por exemplo pacotes UDP destinados a um servidor `syslog` remoto;
- realizar um ataque de negação de serviço (*Denial of Service*), informando um mapeamento para um endereço *ethernet* inválido, como por exemplo o do endereço de um *gateway*;
- encher o *cache* ARP de *bridges* e *switches*, com consequências imprevisíveis.

Para minimizar os problemas acima, é importante que todos os endereços *ethernet* de um determinado segmento sejam conhecidos. Idealmente, todos os *hosts* devem possuir entradas ARP estáticas, o que no caso de um *honeypot* não é possível, pois o invasor tem total controle sobre a máquina. Nesse caso pode ser útil monitorar o tráfego ARP e compará-lo com os resultados conhecidos.

Em *firewalls* que atuem como *bridges* (camada de enlace) e sejam utilizados como mecanismo de contenção de uma *honeynet*, é muito importante que as

entradas em seu *cache* ARP sejam estáticas, previamente preenchidas com os endereços *ethernet* de todos os *honeypots*, e que todo tráfego originado de endereços *ethernet* não conhecidos seja barrado. Esta funcionalidade está disponível, por exemplo, no sistema OpenBSD através do comando `brconfig(8)`, que manipula interfaces do tipo *bridge*.

2.2 Filtragem de Pacotes

Um método simples de contenção de saída, porém bastante efetivo, é a filtragem de certas categorias de pacotes através de regras estáticas implementadas por um *firewall*. Idealmente esse *firewall* pode operar no nível de enlace, como uma *bridge*, diminuindo as chances de ser detectado e atacado pelo invasor.

O tráfego de saída candidato a filtragem, inclui:

- algumas categorias de ICMP, tipicamente usadas por ferramentas de *scan*, como por exemplo ICMP *address mask request*, ICMP *time stamp request* e ICMP *information request*;
- tráfego UDP, com exceção de poucos serviços legítimos, tais como DNS e NTP;
- tráfego TCP com destino a portas associadas a serviços com histórico de vulnerabilidades de segurança, tais como 111/TCP, 515/TCP, entre outros;
- tráfego não usual, como por exemplo pacotes TCP com porta origem igual à porta destino, característica comum no tráfego gerado por muitas ferramentas de *scan*;
- tráfego com IP *options*, como *source routing* por exemplo, que pode ser utilizado por um atacante em ataques de IP *spoofing* [6];
- protocolos[1] não usuais, como por exemplo 0, 53, 55, 77, 103, 255, muitas vezes associados a ataques de negação de serviço;

A grande desvantagem desse método, entretanto, é a sua falta de seletividade para conter as ações de um invasor. Na maioria dos filtros de pacotes, por exemplo, não é possível criar uma regra de filtragem que permita que ele efetue, via FTP, uma transferência de arquivos, mas o impeça de realizar uma grande varredura com destino à porta deste serviço (21/TCP).

2.3 Egress Filtering

A geração de tráfego com endereço IP de origem forjado (*Source Address Spoofing*) pode ser utilizada por um invasor para a execução de várias atividades hostis [7], entre elas:

- algumas categorias de ataques de negação de serviço, como TCP SYN *Flood*, que normalmente dependem que o endereço IP de origem não seja alcançável [8];
- ataques de negação de serviço, como *floods* de UDP ou ICMP, de modo a dificultar a determinação de sua origem;
- algumas categorias de ataques que exploram a relação de confiança entre *hosts* em função dos seus endereços IPs, como por exemplo os "r" *commands* (`rsh`, `rlogin`, etc.);
- *loop* entre serviços, como por exemplo *chargen* e *echo* UDP [9];
- envio de pacotes ICMP ou UDP, destinados a endereços de *broadcast* de diversas redes e com endereços de origem da vítima, gerando uma grande quantidade de respostas direcionadas à vítima, como por exemplo os ataques do tipo "*Smurf*" [10].

A implantação de *Egress Filtering* [11], isto é, permitir apenas a saída de pacotes cujo o endereço IP de origem pertença ao mesmo bloco de endereços designados a uma rede, é uma medida simples mas de especial importância para impedir os abusos citados.

2.4 Limitação de Banda

Um invasor, decidido a utilizar um *host* para lançar ataques de negação de serviço contra outras redes, tipicamente tentará usar toda a banda disponível. Existem diversas ferramentas para este fim, permitindo a geração de um enorme volume de pacotes que pode facilmente saturar uma rede. O tráfego gerado é variável, incluindo protocolos como UDP, ICMP ou outros menos usuais, e pode ou não ter endereço de origem forjado.

Para limitar o impacto desse tipo de ataque é desejável que algum tipo de controle do consumo de banda de saída seja implementado. Uma solução possível é utilizar o sistema ALTQ, um *framework* de manipulação de disciplinas de filas em interfaces de rede. Esse sistema manipula filas de saída de modo a definir limites de consumo de banda e a priorizar tráfego [12].

Alguns filtros de pacotes permitem associar regras de filtragem com limitação de banda. Deste modo é possível criar diferentes filas de saída para diferentes regras, cada uma com um valor de banda diferente, levando em consideração, por exemplo, diferentes protocolos e serviços. Pode-se, por exemplo, criar uma regra de saída específica para SMTP, com um valor de banda muito baixo, que permita a um invasor enviar *emails* mas que torne inviável abusos, como o envio de um grande volume de *spam*.

[1] `http://www.iana.org/assignments/protocol-numbers`

Idealmente a limitação de banda não deve ser a única maneira de conter ataques de negação de serviço contra outras redes, mas sim um mecanismo adicional às demais técnicas aqui discutidas.

2.5 Normalização de Tráfego

Ao explorar a possibilidade de ambigüidades no tráfego de rede, é possível para um invasor lançar um ataque e mesmo assim escapar da detecção de um IDS baseado em rede (NIDS) [13].

A seguir são mostradas algumas situações de tráfego com ambigüidades que podem dificultar a detecção de ataques:

- ocorrência de fragmentos de pacotes IP, no caso de sensores que não realizam remontagem de fragmentos;
- ocorrência de fragmentos de pacotes IP fora de ordem, duplicados ou com sobreposição, quando utilizados sensores que remontam fragmentos, mas o fazem incorretamente nesses casos. Algumas ferramentas permitem que um invasor gere facilmente ataques com essas características, como por exemplo o `fragroute`[2];
- situações não completamente descritas nas especificações dos protocolos, como por exemplo tamanhos não usuais de pacotes, combinação inválida de *flags* TCP, etc.

De forma a impedir que ataques que explorem ambigüidades nos protocolos sejam lançados, ou pelo menos aumentar as chances desses ataques serem detectados nos sistemas destino, utiliza-se um normalizador de tráfego. Este consiste em um elemento de rede que tem por objetivo remover as ambigüidades no tráfego que por ele passa [13]. Exemplos de normalizadores incluem o programa `norm` e a diretiva `scrub` do OpenBSD `pf`.

2.6 Limitação do Número de Sessões

A limitação do número de sessões é de especial importância para conter certas atividades iniciadas por um invasor contra outras redes, como varreduras e certos tipos de ataques de negação de serviço, mas ainda assim permitir um certo nível de atividade de maneira controlada. Esta técnica supre algumas deficiências daquela descrita na seção 2.2.

Para diminuir, por parte do invasor, a suspeita de monitoração, é desejável afetar apenas as sessões de saída, permitindo que as sessões de entrada nas máquinas comprometidas permaneçam inalteradas. Isso pode ser implementado em *firewalls* que priorizem filtragem *stateful* sobre regras estáticas [14].

[2] `http://monkey.org/~dugsong/fragroute/`

2.7 Limitação do Volume de Dados por Sessão

Embora a limitação do número de sessões de saída estabelecidas seja extremamente importante, como por exemplo no controle da execução de varreduras e lançamento de algumas categorias de negação de serviço, nem todos os ataques necessitam da criação de muitas sessões.

Um exemplo são alguns ataques de negação de serviço que utilizam *floods* ICMP[3] — onde é criada uma única sessão, por onde passa todo o fluxo de dados de saída do invasor.

Nesses casos é desejável impor um limite máximo de *bytes* enviados e, quando esse limite for alcançado, interromper a sessão em andamento e colocar uma regra de filtragem. É possível ainda utilizar algum mecanismo de limitação de banda (seção 2.4), de modo a diminuir progressivamente a banda disponível assim que um determinado limite de *bytes* enviados seja alcançado.

2.8 Alteração Dinâmica de Regras de Filtragem

Uma solução para o problema da falta de seletividade de alguns filtros de pacotes, como descrito na seção 2.2, é a alteração dinâmica das regras de saída do *firewall*.

De especial interesse, para uma aplicação que tenha o objetivo de monitorar tráfego para geração de regras dinâmicas, são as seguintes características:

- número máximo de sessões de saída estabelecidas, por *host* de origem;
- elevada taxa de crescimento do número de sessões de saída criadas por um mesmo *host*;
- volume máximo de dados transferidos em uma sessão;
- tempo máximo de conexão de uma sessão.

Com base nestas características, a aplicação pode interagir com um *firewall*, interrompendo sessões em andamento e inserindo novas regras de filtragem, conforme necessário. Também é possível que a aplicação remova regras de filtragem após um certo período de tempo.

2.9 Filtragem por Conteúdo de Pacotes

Assim como é possível para um invasor explorar uma vulnerabilidade em uma máquina remota sem a criação de muitas sessões, como discutido na seção 2.7, também é possível executar ataques bem sucedidos sem o envio de um volume exagerado de dados.

Para barrar ataques com estas características, é desejável um mecanismo que torne possível descartar

[3] Como por exemplo um trivial "`ping -f vitima`", que gera um *flood* de ICMP *echo request* sobre a máquina vítima.

pacotes de saída em função de padrões conhecidos de ataques presentes no seu conteúdo.

Para implementar esse tipo de contenção podem ser usadas ferramentas como o `snort-inline`[4] e o `hogwash`[5], que possuem uma grande coleção de assinaturas de ataques conhecidos.

Entretanto, esse tipo de contenção de tráfego possui alguns problemas:

- é limitado apenas a padrões de ataques conhecidos;
- é normalmente lento, pois envolve a cópia de um pacote que está em um espaço de endereçamento reservado para o sistema operacional (*kernel space*), para uma área de memória alocada para a aplicação (*user space*);
- identificar padrões de ataques pode ser complicado por fatores como fragmentação de pacotes, como discutido na seção 2.5.

2.10 Modificação de Pacotes

Além da filtragem de pacotes de saída em função do seu conteúdo, outra possibilidade consiste na modificação desses pacotes.

A modificação de alguns *bytes* do conteúdo de um pacote é eficaz para impedir o funcionamento esperado de certos ataques. Esta modificação pode ser bastante sutil e de difícil detecção por parte do invasor, que pode ser levado a acreditar que a máquina alvo não é suscetível a uma determinada vulnerabilidade.

Outra possibilidade é alterar os cabeçalhos dos pacotes, de modo que roteadores entre o invasor e a vítima os descartem. Exemplos são alterações do *checksum* do cabeçalho IP e TCP, diminuição do valor de TTL (*Time to Live*), etc.

2.11 Redirecionamento de Tráfego

Outra possibilidade de contenção consiste na redireção de pacotes de saída. Os pacote podem ser redirecionados para uma máquina destino diferente da original e, deste modo, além de proteger a máquina vítima, fazer com que o invasor acredite que seu ataque teve sucesso.

3 IMPLEMENTAÇÃO

Esta seção descreve a implementação de uma ferramenta de código aberto, `sessionlimit`, que implementa as técnicas de limitação do número de sessões, limitação do volume de dados por sessão e alteração dinâmica de regras de filtragem, descritas na seção 2. Um protótipo desta ferramenta foi inicialmente escrito em Perl, mas por questões de desempenho, as versões posteriores foram escritas em linguagem C.

[4] `http://www.snort.org/dl/contrib/patches/inline/`
[5] `http://hogwash.sourceforge.net/`

Esta ferramenta foi escrita para operar em conjunto com o filtro de pacotes `pf`, do sistema operacional OpenBSD, e foi desenvolvida no âmbito do Projeto Honeynet.BR. Ela tem sido um importante mecanismo de contenção de tráfego malicioso de saída neste projeto [3].

A implementação desta ferramenta baseou-se fortemente nas características de filtros de pacotes *stateful*. Um filtro de pacotes *stateful* não inspeciona apenas pacotes isoladamente, mas também possui o conceito de sessões estabelecidas. Nele, qualquer regra de filtragem que permita a passagem de um pacote pode ou não criar uma entrada na tabela de estados. Antes que o conjunto de regras de filtragem seja consultado para um dado pacote, a tabela de estados é consultada — se o pacote fizer parte de uma sessão estabelecida, é passado incondicionalmente, sem necessidade de consultar as regras de filtragem.

O filtro de pacotes `pf` armazena as entradas da tabela de estados numa árvore AVL, de modo que buscas nessa árvore são $O(\log m)$, onde m é o número de estados. Isso é mais eficiente do que uma busca nas regras de filtragem, que é $O(n)$, onde n é o número de regras. Desse modo, a filtragem *stateful* não apenas aumenta a qualidade das decisões de filtragem, como melhora também o seu desempenho [14].

A seguir será descrito o funcionamento desta ferramenta.

3.1 Monitoração da Tabela de Estados

O programa `sessionlimit` opera em um *firewall* executando o filtro de pacotes `pf`, configurado para criar entradas na sua tabela de estados para pacotes de saída.

Consultando a tabela de estados do `pf` a um dado instante, é possível obter várias informações sobre as sessões em andamento, entre elas:

- endereço IP de origem e destino;
- direção da sessão (de entrada ou de saída);
- regra de filtragem que inicialmente criou o estado;
- data de criação e expiração da sessão;
- número de pacotes e *bytes* enviados;
- protocolo;
- *status* (`ESTABLISHED`, `FIN_WAIT`, etc.).

A tabela de estados é mantida no *kernel* do sistema operacional e é formada por um conjunto de estruturas do tipo `pf_state`, onde estão armazenadas todas as informações sobre cada estado criado. A definição desta estrutura é mostrada na Fig. 1.

Em sistemas Unix em geral, a chamada de sistema `ioctl(2)` é utilizada para realizar operações de

```
struct pf_state {
        struct pf_state_host lan;
        struct pf_state_host gwy;
        struct pf_state_host ext;
        struct pf_state_peer src;
        struct pf_state_peer dst;
        union {
                struct pf_rule  *ptr;
                u_int32_t        nr;
        } rule;
        struct pf_rule  *nat_rule;
        struct pf_addr   rt_addr;
        struct ifnet    *rt_ifp;
        u_int32_t        creation;
        u_int32_t        expire;
        u_int32_t        packets;
        u_int32_t        bytes;
        sa_family_t      af;
        u_int8_t         proto;
        u_int8_t         direction;
        u_int8_t         log;
        u_int8_t         allow_opts;
        u_int8_t         pad[3];
};
```

Figura 1: Estrutura `pf_state`, usada para armazenar as informações de cada estado na tabela de estados do *kernel*.

controle que são específicas de determinados dispositivos [15], como por exemplo terminais e *sockets*. No sistema OpenBSD, utilizando a chamada de sistema `ioctl(2)` e o pseudo dispositivo `/dev/pf`, um programa em *user space* pode interagir com o filtro de pacotes, inserindo e removendo regras e fazendo consultas sobre a tabela de estados.

O programa `sessionlimit` monitora as sessões de saída na tabela de estados do sistema, de modo a identificar tráfego que atenda a um dos critérios abaixo:

- taxa de crescimento muito rápida no número de estados associados a um endereço IP de origem, atividade que é tipicamente associada a varreduras;
- número máximo de estados associados a um determinado endereço IP de origem;
- número máximo de *bytes* transmitidos, associados a uma sessão ICMP.

Caso alguma dessas condições seja satisfeita por um *host*, este é identificado para ter seu tráfego bloqueado.

A decisão pelo bloqueio de um *host* com base nestas condições permite conter, por exemplo, varreduras a partir da *honeynet*, mas não gera regras de bloqueio em caso de uso da rede pelo invasor para realizar atividades de interesse para *honeynets*, como por exemplo transferências de ferramentas via FTP/HTTP ou sessões de IRC [3].

3.2 Inserção e Remoção de Regras de Filtragem

De maneira análoga à consulta à tabela de estados, a chamada de sistema `ioctl(2)` também é utilizada para a inserção e remoção de regras de filtragem.

Uma vez que um *host* satisfaça os critérios descritos na seção anterior, uma regra bloqueando o tráfego deste *host* é inserida nas regras correntes do `pf` e as sessões de saída deste IP, em andamento, são removidas. Um exemplo de uma regra inserida pelo `sessionlimit` é mostrada na Fig. 2.

É importante notar que a regra de bloqueio inserida pelo `sessionlimit` afeta apenas o tráfego de saída. As sessões de entrada já estabelecidas para o *host*, que tipicamente incluem a sessão interativa do invasor, não são afetadas.

Uma regra de bloqueio expira após um certo tempo[6], sendo então retirada pelo `sessionlimit` da lista de regras ativas do *firewall*.

As ações tomadas pelo `sessionlimit` são registradas via o mecanismo de `syslog(3)` [15], como mostrado na Fig. 2.

```
block drop out log quick on int0 inet \
from 192.0.2.1 to any label "sessionlimit"

[ Evaluations: 136185    Packets: 67292 \
  Bytes: 3230906     States: 0     ]
```

```
Jun 20 15:10:25 host sessionlimit[6761]: \
    blocking 192.0.2.1 (20 states)

Jun 20 15:10:25 host sessionlimit[6761]: \
    21 state(s) killed from 192.0.2.1
```

Figura 2: Regra de *firewall* inserida pelo `sessionlimit` e trechos de *log* gerados. Algumas linhas foram quebradas por questões de legibilidade. Os IPs reais, *hostnames* e nome de *interface* foram sanitizados.

4 CONCLUSÕES E TRABALHOS FUTUROS

Este artigo descreveu algumas categorias de tráfego malicioso de saída que podem estar associadas a ações de invasores e algumas técnicas para conter esse tráfego. Foi também apresentada a implementação de uma ferramenta, que aplica alguns desses métodos, para ser utilizada na contenção de tráfego de saída em *honeynets*.

Esta ferramenta tem se mostrado bastante estável e eficiente na contenção de várias categorias de tráfego malicioso, porém possui como uma das limitações o fato de consumir mais CPU do que o desejável no seu processo de monitoramento da tabela de estados do *firewall*. Estão sendo estudados métodos para melhorar o desempenho do programa nesse aspecto.

[6] Atualmente 30 minutos, mas esse valor é configurável.

Com relação às técnicas atuais de contenção suportadas, planeja-se adicionar a filtragem por conteúdo de pacotes e integrá-la com a limitação do número de sessões e volume de dados.

Pretende-se também tornar a configuração da ferramenta mais flexível, com a possibilidade de parâmetros distintos para cada um dos *hosts* monitorados.

AGRADECIMENTOS

Os autores gostariam de agradecer aos membros da Honeynet.BR, pelas suas idéias e ajuda em testar o `sessionlimit`.

REFERÊNCIAS

[1] L. Spitzner, "Learning the Tools and the Tactics of the Enemy with Honeynets," in *Proceedings of the 12th Annual Computer Security Incident Handling Conference*, (Chicago, IL, USA), June 2000.

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

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

[4] L. Spitzner, *Honeypots: Tracking Hackers*. Addison-Wesley, 1st ed., September 2002. ISBN 0-321-10895-7.

[5] D. C. Plummer, "RFC 826: An Ethernet Address Resolution Protocol: or Converting Network Protocol Addresses to 48.bit Ethernet Address for Transmission on Ethernet Hardware." `http://www.ietf.org/rfc/rfc826.txt`, 1982 November.

[6] S. M. Bellovin, "Security Problems in the TCP/IP Protocol Suite," *Computer Communications Review*, vol. 19, pp. 32–48, April 1989.

[7] P. Ferguson and D. Senie, "RFC 2827: Network Ingress Filtering: Defeating Denial of Service Attacks which employ IP Source Address Spoofing." `http://www.ietf.org/rfc/rfc2827.txt`, 2000 May.

[8] CERT Coordination Center, "CERT Advisory CA-1996-21 TCP SYN Flooding and IP Spoofing Attacks." `http://www.cert.org/advisories/CA-1996-21.html`, 1996.

[9] CERT Coordination Center, "CERT Advisory CA-1996-01 UDP Port Denial-of-Service Attack." `http://www.cert.org/advisories/CA-1996-01.html`, 1996.

[10] CERT Coordination Center, "CERT Advisory CA-1998-01 Smurf IP Denial-of-Service Attacks." `http://www.cert.org/advisories/CA-1998-01.html`, 1998.

[11] T. Killalea, "Recommended Internet Service Provider Security Services and Procedures." `http://www.ietf.org/rfc/rfc3013.txt`, 2000 November.

[12] K. Cho, "Managing Traffic with ALTQ," in *Proceedings of the FREENIX Track: 1999 USENIX Annual Technical Conference (FREENIX '99)*, (Monterey, CA, USA), June 1999.

[13] M. Handley, V. Paxson, and C. Kreibich, "Network Intrusion Detection: Evasion, Traffic Normalization, and End-to-End Protocol Semantics," in *Proceedings of the 10th USENIX Security Symposium*, (Washington, D.C.), August 2001.

[14] D. Hartmeier, "Design and Performance of the OpenBSD Stateful Packet Filter (pf)," in *Proceedings of the FREENIX Track: 2002 USENIX Annual Technical Conference (FREENIX '02)*, (Monterey, CA, USA), June 2002.

[15] W. R. Stevens, *Advanced Programming in the UNIX Environment*. Addison-Wesley, 1st ed., December 1994. ISBN 0-201-56317-7.