Towards Secure and Reliable Firewall Systems based on Minix 3

Rüdiger Weis, Brian Schüler, Stefan Flemming*

Beuth Hochschule für Technik Berlin, University of Applied Sciences
{rcw,bschueler,flemming}@bht-berlin.de

Abstract: Minix 3 is a real micro kernel operation system with a lot of remarkable security features. Two of the main points are size and isolation. The Minix 3 kernel is less than one thousand times the size of Linux. All drivers and the IP stack live in user land. We show a port of the netfilter framework, which leads to a system with better stability and security than the widely used Linux solutions [We07]. Additionally we present some new ideas regarding virtualized systems.

1 Introduction

"your job is being a professor and researcher: That's one hell of a good excuse for some of the brain-damages of minix", Linus Torwalds, 1992.

"I mean, sometimes it's a bit sad and we're definitely not the streamlined hyper-efficient kernel that I had envisioned 15 years ago. The kernel is huge and bloated", Linus Torwalds, 2009.

Security flaws in modern Operating Systems

The security problems facing the current crop of operating systems, including Windows, but also including Linux, are the result of design errors. The errors were inherited for the most part from their predecessors of the 1960s.

Most of these problems can be attributed to the fact that developers aren't perfect. Humans make mistakes. Of course, it would be nice to reduce the numbers and mitigate the effects; however, designers have frequently been far too willing to compromise security and a clean design for speed. Tanenbaum refers to this as a 'Faustian pact'.

In addition to the issues related to sheer size, monolithic designs are also prone to inherent structural problems: Any error is capable of endangering the whole system. A fundamental design error is that current operating systems do not follow the Principle Of Least

*Supported by the Projekt Forschungsassistenz from Europäischer Sozialfonds.

Authority (POLA). To put this simply, POLA states that developers should distribute systems over a number of modules so an error in one module will not compromise the security and stability of other modules. They should also make sure that each module only has the rights that it actually needs to complete its assigned tasks.

The Minix way

Minix is often viewed as the spiritual predecessor of Linux, but these two Unix cousins could never agree on the kernel design. Minix 3 has been designed to reduce the lines of kernel code. With less than 5.000 loc the Minix 3 kernel is less than one thousand times the size of the Linux kernel.

Minix 3 runs on 32-bit x86 CPUs, as well as on a number of virtual machines including Qemu, Xen, and VMware. The operating system includes an X Window System (X11), a number of editors (Emacs and Vi), shells (including bash and Zsh), GCC, script languages such as Python and Perl, and network tools such as SSH. A small footprint and crash-resistant design make Minix a good candidate for embedded systems, and its superior stability has led to a promising new role as a firewall system.

2 The MinixWall

Packet filters are an endangered system component. Despite the pretty good quality of the Linux Netfilter implementation, a number of security issues have surfaced in the past. If a subsystem of this kind is running on the Linux kernel, it will endanger system security. Building on work by the Tanenbaum group, the Technical University of Applied Science Berlin ported the widespread Netfilter framework to Minix 3.

In Minix 3, a single user process could crash without compromising system security. In most cases the reincarnation server would simply restart the process. The differences become even more apparent if an attacker succeeds in executing code. In Minix, a hijacked user process is still a problem, but the effect is far less serious thanks to isolation.

The fact that security-relevant programs such as network packet filters are executed directly inside the kernel space intensifies the problem. This is the cause why a vulnerability in the firewall code does not only effect the filter routines but also influences the whole Linux kernel. A crash of the process thereby not only has an impact on the overall system stability but can be also used in attacks to circumvent system security mechanisms.

2.1 Operating a firewall in Linux and Minix systems

Due to the highly security enhanced and stable design, Minix is predestinated to be run in security critical systems such as firewalls. Figure 1 shows the operation of a packet

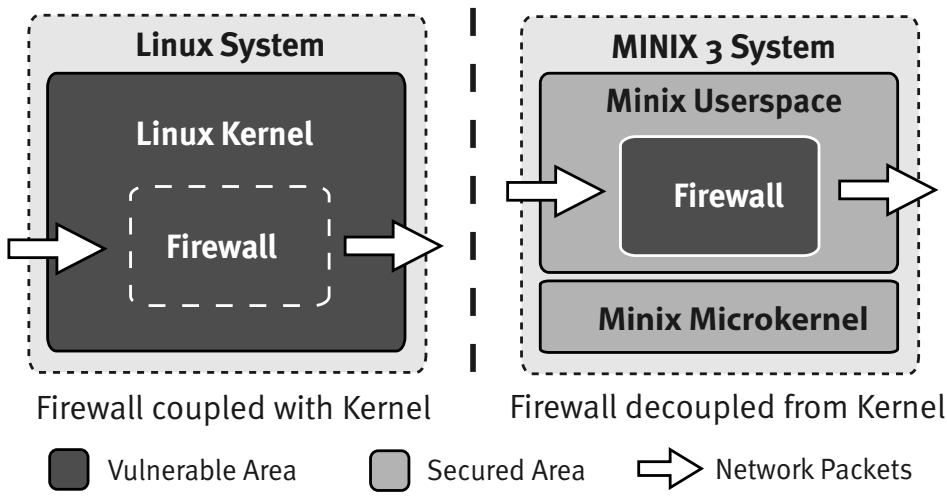

Figure 1: Vulnerability comparison - Linux versus Minix

filter on Linux, compared to one running on Minix. While faulty firewall code in Linux is directly executed inside the kernel space, an exploit will result in an unstable system or even a kernel panic. Beyond that it can be used to hijack the whole kernel. In Minix the execution inside the userspace effectively separates the firewall code from the rest of the system. A crash caused by the packet filter will so only effect the userspace process itself and neither the kernel nor other processes.

2.2 MinixWall - A secure packet filter for Minix

In 2007 the project MinixWall [Sc07] started at the Beuth University of Applied Sciences Berlin. MinixWall ist a port of the iptables/netfilter firewall framework, which is normally part of the Linux kernel. All required sources have been taken from the Linux kernel and freed from Linux-specific structures to make it more portable.

On the Minix side, the network stack daemon *inet* required to be modified by an additional interface that is responsible for the redirection of all network packets to the network filter process. This filter is part of a newly introduced sub-layer for packet filtering tasks. While the filter functions in Linux are executed as sub-routines within the TCP/IP stack, the *MinixWall* filters network packages by a single filter process in user mode and enables to transport them using a device file. In Linux an exploit in the filter is executed inside a sub-routine of the TCP/IP stack and will provoke a kernel panic. A security leak in Minix on the other hand will only crash the single user process without compromising the system stability. In many cases the restart of the service by the *reincarnation server* brings the system again into working state.

To ease the configuration of the netfilter, the userspace application *iptables*, which is well known on Linux environments has been included into the MinixWall distribution in forms of a separate application. This helper tool uses simple device control sequences (ioctl) to configure the firewall rule information. The corresponding filter device is published by */dev/netfilter* and can be made accessible to a specific user and a group that is allowed to administrate the rule set.

Two additional sets of functions have been implemented to deal with the Minix network stack, the *inet* daemon.

- The first set of functions are for exchanging network packet data between the *inet* daemon and the firewall process and are called the backend functions.
  - The functions `inetEthIn(char *ethname)` and `inetEthOut` are for telling the filter which ethernet input and/or output devices are concerned for a network packet (i.e. IN=eth0 and OUT=eth1 in the case of a forwarded packet). One of these functions must be called to initialize the filter state machine that is responsible for the different network packet headers.
  - The functions `inetSetPackSize()` and `inetSetDataSize()` are used to define the total packet size and the data size of the upcoming header respectively.
  - The `inetData()` function transfers a buffer filled with the network packet data to the firewall to be processed later with `inetProcess()`.
- The second set of functions are frontend functions and are designed for setting up and editing the firewall rule set.
  - The command line tool *iptables* from Linux is used here as a frontend. A new filter table is created with `iptablesNewChain(...)` function for example.
  - `iptablesSelectTable(...)` and `iptablesSelectChain(...)` are used for selecting tables and chains.
  - Rules can be appended with `iptablesAppendRule()` to an existing chain.

All these comfortable backend and frontend functions are translated into *ioctl's* sequences that are used for the firewalling device file. The header files which MinixWall uses are similar to these on Linux.

The following example code in the *inet* daemon is passed when a network packet arrives the local machine and performs a full filtering in the INPUT direction:

Protocol	Linux include file	Minix include file
IP header	/usr/include/linux/ip.h	/usr/include/net/gen/ip_hdr.h
ICMP header	/usr/include/net/icmp.h	/usr/include/net/gen/icmp_hdr.h
UDP header	/usr/include/net/udp.h	/usr/include/net/gen/udp_hdr.h
TCP header	/usr/include/net/tcp.h	/usr/include/net/gen/tcp_hdr.h
Routing information	/usr/include/net/route.h	/usr/include/net/gen/route.h

Table 1: networking include files

3 Virtualization

A Minix based system is a reliable and secure platform for the operation of firewalls, but often it is meaningful to take full advantage of the hardware by hosting additional network services on the same machine. Operating all these services directly under Minix already offers several key benefits compared to other operating systems that are not as stable, secure and resource-friendly. Whenever a service crashes, it will only affect one userspace server process that will be restarted immediately.

The classical way to deal with these problems is to operate the *MinixWall* on a standalone network host and forward all filtered traffic to a second computer that is running a different operating system with all required network services such as a mail-, web-, ftp- or file-server. This topology clearly separates the systems with the advantage that exploits do not effect more than one machine but also the disadvantage that additional hardware is required.

3.1 Operation of complete networks on one hardware

An interesting approach is the combination of a Minix 3 based *MinixWall* with other operating systems using virtualization. As shown in figure 2, the hardware of a router is hosting two independent virtualized guest systems. One guest can so operate the *MinixWall* packet filter while the other one serves additional network services. An important advantage of this approach is the need for only one computer while both operating systems are kept separated. Beyond that the user is free to select the best fitting operating system for a specific task.

In respect to the security and stability it is apparent that the virtualized design has advantages caused by this separation. Whenever one guest fails, it can be restarted independently from other systems. If one guest is compromised by an attack, it can be replaced using an image without affecting the other ones. This approach not only enables the separation of critical services in its own virtualization containers but also offers the operation of virtualized guests with different operating systems at the same time. Several combinations of servers and virtualized firewalls are conceivable such as complete networks with DMZs and multiple differently filtered subnets.

```
nf_ioctl(IOCTL_NF_ETH_IN,(void*)ifin);
nf_ioctl(IOCTL_NF_ETH_OUT,(void*)ifout);
nf_ioctl(IOCTL_NF_PACK_SIZE,(void*)pack_size);
nf_ioctl(IOCTL_NF_CONTAINS,(void*)NF_LAYER_IP);
/* sending all buffer parts to the filter */
for (i=0, temppack=pack, count=pack_size; temppack && count>0;)
{
nf_ioctl(IOCTL_NF_SIZE,(void*)temppack->acc_buffer->buf_size);
nf_ioctl(IOCTL_NF_DATA,
(void*) (((vir_bytes)temppack->acc_buffer->buf_data_p)
+temppack->acc_offset)
);
i++; count-=temppack->acc_buffer->buf_size;
temppack=temppack->acc_next;
}
nf_ioctl(IOCTL_NF_HOOK,(void*)NF_IP_LOCAL_IN);

/* do the filtering work */
if (nf_ioctl(IOCTL_NF_PROCESS,NULL)==1)
{
if (broadcast)
ip_arrived_broadcast(ip_port, pack);
else
ip_arrived(ip_port, pack);
}
nf_ioctl(IOCTL_NF_GETDATA,NULL);
```

Table 2: example filter code

3.2 Cascaded firewalls

Figure 2 illustrates that all incoming and unfiltered traffic is forwarded to one *MinixWall*, when it enters the host system. This traffic is guided through the *MinixWall* to be filtered by the packet filter before it is redirected to the Linux guestsystem that acts as a host for all services.

The other way around the packages generated by the *Linux* guest for the outside world can be filtered again by the *MinixWall*, before they leave the router host.

For all router scenarios Minix is equipped with two virtual network interface cards for the internal and external network traffic. One card is directly bridged to the network interface card of the host system while the other one is used for the internal traffic. The hostsystem itself does not require more than one network interface card but can be equipped with a second card if the filtered traffic should be made available to an external system.

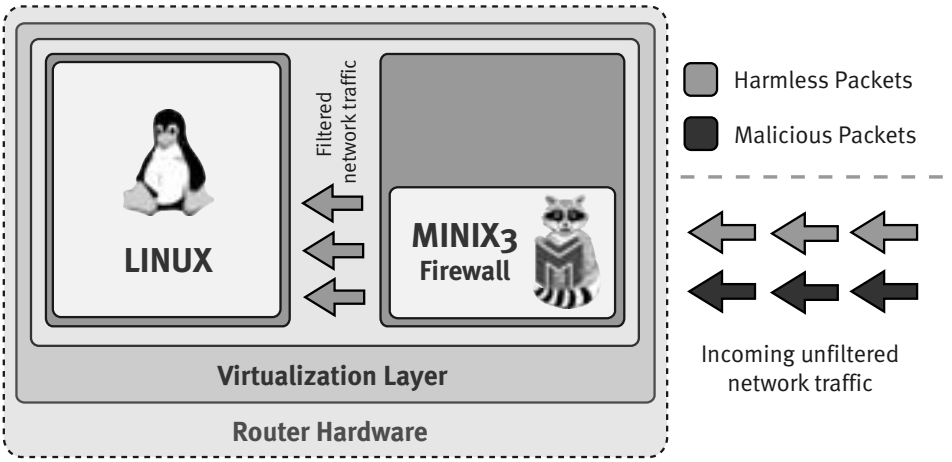

Figure 2: Virtualization of the Minix Packetfilter

While figure 2 shows a solution with only one firewall, it can be extended by additional packet filters to a cascaded firewall solution. Multiple virtualized firewalls can so filter traffic in series to provide redundant security. The risk that an attacker is able to compromise the system is so efficiently reduced, because an attacker needs to break all involved firewalls.

4 Conclusion and Outlook

We have presented a firewall system based on Minix 3 which provides higher security and better reliability by design. Combined with the very low hardware requirements this system has a wide range of applications in the field of embedded hardware and on virtualized systems. It is clear that security researchers would prefer putting the firewall systems onto dedicated hardware, but our solution can be used without additional hardware costs in many scenarios. We are also working on improvements for virtualized networks on one system (e.g. DMZ). All our software is published under der GNU Public Licsence and can be downloaded from: `http://wiki.beuth-hochschule.de/~minixwall`.

References

[Mi05] Minix 3: *The official Minix3 web page*. http://www.minix3.org.

[He06] Herder, Jorrit N.: *TOWARDS A TRUE MICROKERNEL OPERATING SYSTEM*. http://www.minix3.org/doc/herder_thesis.pdf, 2006.

[Sc07] Schüler, Brian : *Analysis and Porting of a network filtering architecture on Minix-3*, Diplomarbeit, TFH Berlin, 2007.

[We07] Weis, Rüdiger : *Linux is obsolete 2.0*, CCCamp 2007, http://public.beuth-hochschule.de/~rweis/vorlesungen/ComputerSicherheit/WeisLinuxIsObsolete2.pdf, 2007.

[We09a] Weis, Rüdiger : *Smart Kernel*, Linux Magazine, 2009. http://www.linux-magazine.com/w3/issue/99/Minix_3_Review.pdf

[We09b] Weis, Rüdiger : *Mit Schutzmantel*, Linux Magazin, 2009. http://www.linux-magazin.de/Heft-Abo/Ausgaben/2009/01/Mit-Schutzmantel?category=0

[Ke06] Kelly, Ivan: *Final Year Project*. Porting Minix to Xen, 2006.

[Ta06] Tanenbaum, Andrew. S. : *Introduction to Minix3*. http://www.osnews.com/story/15960/Introduction-to-Minix-3, 2006.

[He06] Herder, J.N. and Bos, H. and Gras, B. and Homburg, P. and Tanenbaum, A.S.: *Minix 3: A highly reliable, self-repairing operating system*. ACM SIGOPS Operating Systems Review, 2006.

[THB06] Tanenbaum, AS and Herder, JN and Bos, H.: *Can we make operating systems reliable and secure?*. Computer Journal Volume 39 Number 5, 2006.

[He07] Herder, J. and Bos, H. and Gras, B. and Homburg, P. and Tanenbaum, A.S.: *Roadmap to a failure-resilient operating system*. https://www.usenix.org/publications/login/2007-02/openpdfs/herder.pdf, 2006.

[Ko09] Van der Kouwe, E.: *Porting the QEMU virtualization software to Minix 3*, 2009.

This work has been supported by the Projekt Forschungsassistenz from Europäischer Sozialfonds (ESF) and the Senat of Berlin.