University of Amsterdam

System and Network Engineering

Offensive Technologies

Looking back at Grsecurity

Authors:
Eddie Bijnen, student
Mike Berkelaar, student

June 1, 2014

Abstract

Grsecurity is collection of kernel modifications that aim to improve the security of the Linux operating system. The approach of Grsecurity is to take out entire classes of exploits. This is done by modifying the kernel to make each of the methods of attack impossible. Because of this approach a level of zero-day protection is claimed.

In this report we investigate these claims. We have gathered an array of potentially system compromising exploits that have been discovered in a five year time frame. Each of these exploits are tested on a vulnerable system that has yet to receive a fix for the exploit, simulating a zero-day attack. We found that Grsecurity offers considerable enhancements in the field of kernel security and that from a historical point of view a lot of zero-day exploits are indeed mitigated.

Contents

List of Tables

1 Introduction

Grsecurity is a security enhancement of the Linux kernel that is not included by default. It tries to mitigate exploits that utilize low-level vulnerabilities that are present in the Linux kernel or running software. It's approach is to prevent exploits from being successful, instead of solving the bug after becoming aware of the exploit, a certain level of zero-day resilience is claimed.[4] Although Grsecurity is a well respected security project, we found it is rarely used or implemented by the popular Linux distributions of today.
Measuring the performance of Grsecurity against currently unknown exploits and attacks is not possible. It is however possible to measure the effectiveness of the features of Grsecurity against exploits that were unknown at the time of historical Grsecurity releases.

1.1 Approach

To evaluate the claims of Grsecurity to protect against zero day exploits we will create a system with the state of 5 years ago, containing a substantial amount of exploits publicly known today. By testing a set of exploits on a system that is protected by Grsecurity and a system without we can compare the results and measure the effectiveness of the protection offered by Grsecurity. This benchmark can be used to make an approximation of the effectiveness of the current version against 0-day exploits.

1.2 Related work

Previous research that has been done into the the effectiveness of Grsecurity have mainly taken theoretical approaches to verify and compare the implemented security. The Case Study Comparing Linux Security Kernel Enhancements [3] has verified the theoretical security. A comparison between Grsecurity and SElinux was performed, including the performance impact, although these results are severely dated (2003) and might not cover the development of Grsecurity of the past 5 years.

2 Research question

This paper will focus on the following research question:

- *How effective is Grsecurity against zero-day exploits?*

  With the following sub-question we try to find potential improvements or points of attention:

- *What is the reason for possible remaining vulnerabilities?*

3 Background

Grsecurity is a collection of kernel modifications that aim to improve the security of the Linux kernel. The modifications can be roughly divided into 4 subsections; PaX, role-based access control, chroot restrictions and miscellaneous protections without a clear category.[5] Each sections helps to protect the system by limiting the access to resources within its scope.

3.1 Role-based access control

Grsecurity's role-based access control offers the ability to limit the available resources per program to the absolute minimum. If a process gets compromised the attacker will only have access to the resources that are necessary for the process to run. This will leave access to system controls separated from the compromised resource. The attacker will be required to use a separate exploit to gain access to additional system resources.[5]

3.2 PaX

PaX is a kernel modification that tags memory into two categories: non-writable executable program memory and non-executable data memory. In addition PaX applies address space layout randomization (ASLR), introducing randomness into the address space of processes and the kernel. The implementation of ASLR with a x86 PaX hardened kernel adds roughly 24 bits of entropy to the address space on every start of a process and the kernel, while only recent Linux kernels provide 12 bits in selected areas. [10]

By setting memory to either non-writable for programme code, or non-executable for data code, buffer overflows become far less potent. Programme code can not be injected into from another process and any buffer overflow in the data section will not be executed by an process outside the original. This, coupled with randomization of memory locations, makes exploiting through vulnerabilities in memory management far more difficult.

3.3 Chroot restrictions

Chroot is a technique that is used to change the root directory for a specific process to a custom directory. Originally not intended for security purpose it has little protection and is relatively easily broken out of. Grsecurity aims to lockdown the chroot to ensure that processes that run inside do not gain access to original system. [5] This is reached by by not allowing the commands in table 1 from inside the chroot to the original system.

No shared memory outside of chroot	No kill outside of chroot
No ptrace outside of chroot	No capget outside of chroot
No setpgid outside of chroot	No getpgid outside of chroot
No getsid outside of chroot	No sending of signals by fcntl
No sysctl writes	No mounting or remounting
No pivot_root	No double chroot
No fchdir out of chroot	Enforced chdir("/") upon chroot
No (f)chmod +s	No mknod
No raising of scheduler priority	Removal of harmful privileges via cap
No connecting to abstract unix domain sockets outside of chroot	No viewing of any process outside of chroot, even if /proc is mounted

Table 1: Chroot modules

3.4 Miscellaneous

Besides the three major components Grsecurity also sets up various additional security measures like removing attack vectors and responding to behavior that may hint on malicious activity. These measures aim to lock down system resources that should rarely be required with normal use. The following list gathers the miscellaneous protections as per [5], each preventing a possible attack or weakness in the Linux system.

- FIFO restrictions
- Dmesg(8) restriction
- GID-based socket restrictions
- Automatic deterrence of exploit brute-forcing
- Tunable flood-time and burst for logging
- Enhanced implementation of Trusted Path Execution
- Symlink/hardlink restrictions to prevent /tmp races
- Nearly all options are sysctl-tunable, with a locking mechanism
- /proc restrictions that don't leak information about process owners
- Detection of local connections: copies attacker's IP address to the other task
- All alerts and audits support a feature that logs the IP address of the attacker with the log
- Stream connections across unix domain sockets carry the attacker's IP address with them (on 2.4 only)

3.5 Security profiles

Configuring Grsecurity can be a complicated task without explicit knowledge of the certain protections Grsecurity offers and what consequences they may have in the form of false-positives. There are more than a dozen options to enable and disable specific protection modules. To help simplify this task there are three predefined security profiles that can be used at build-time of the hardened kernel: *Low, Medium and High.*[5] These profiles come with a preset of enabled modules, starting with the least intrusive and becoming more secure. A higher security profile leads to better security. However, it also leads to less software compatibility and potentially lower performance. The different security levels and their configurations are ordered in table 2 in order to map the security offered by the presets provided.

We can derive from the table that the *Low* preset only offers the most basic functionalities that Grsecurity has in the area of resource starvation and denial of service prevention. The *Medium* preset adds Chroot protections and basic memory security features like ASLR. Only *High* has the most advanced memory security features enabled like execution prevention of writable memory and memory regions outside of the kernel's memory space.

Modules	Low	Medium	High
GRKERNSEC_DMESG	X	X	X
GRKERNSEC_EXECVE	X	X	X
GRKERNSEC_FIFO	X	X	X
GRKERNSEC_LINK	X	X	X
GRKERNSEC_MODSTOP	X	X	X
GRKERNSEC_RANDNET	X	X	X
GRKERNSEC_CHROOT_CHDIR	X	X	X
GRKERNSEC_CHROOT		X	X
GRKERNSEC_CHROOT_DOUBLE		X	X
GRKERNSEC_CHROOT_MKNOD		X	X
GRKERNSEC_CHROOT_MOUNT		X	X
GRKERNSEC_CHROOT_PIVOT		X	X
GRKERNSEC_CHROOT_SYSCTL		X	X
GRKERNSEC_CHROOT_UNIX		X	X
GRKERNSEC_FORKFAIL		X	X
GRKERNSEC_PROC		X	X
GRKERNSEC_PROC_MEMMAP		X	X
GRKERNSEC_PROC_USERGROUP		X	X
GRKERNSEC_SIGNAL		X	X
GRKERNSEC_TIME		X	X
PAX		X	X
PAX_ASLR		X	X
PAX_EI_PAX		X	X
PAX_HAVE_ACL_FLAGS		X	X
PAX_PT_PAX_FLAGS		X	X
PAX_RANDMMAP		X	X
PAX_RANDUSTACK		X	X
PAX_REFCOUNT		X	X
GRKERNSEC_AUDIT_MOUNT			X
GRKERNSEC_BRUTE			X
GRKERNSEC_CHROOT_CAPS			X
GRKERNSEC_CHROOT_CHMOD			X
GRKERNSEC_CHROOT_FCHDIR			X
GRKERNSEC_CHROOT_FINDTASK			X
GRKERNSEC_CHROOT_NICE			X
GRKERNSEC_CHROOT_SHMAT			X
GRKERNSEC_HIDESYM			X
GRKERNSEC_KMEM			X
GRKERNSEC_PROC_ADD			X
GRKERNSEC_RESLOG			X
PAX_DLRESOLVE			X
PAX_ETEXECRELOCS			X
PAX_EMUPLT			X
PAX_EMUTRAMP			X
PAX_KERNEXEC			X
PAX_MEMORY_UDEREF			X
PAX_MPROTECT			X
PAX_NOEXEC			X
PAX_PAGEEXEC			X
PAX_RANDKSTACK			X
PAX_SEGMEXEC			X
PAX_SYSCALL			X

Table 2: Grsecurity modules

4 Methodology

The researchers have chosen to take a hands on approach to verify the claims of Grsecurity and to see if the theoretical security keeps up in practice. To achieve this we have created a subset of virtual machines with known vulnerabilities. The Debian Lenny (5.0.1) release was selected as the target machine as it contains reasonably recent, yet out-of-date, software. To test the vulnerabilities we will utilize exploit concepts that have been published to target certain vulnerabilities as detailed in corresponding CVEs (Common Vulnerabilities and Exposures), a database that gathers publicly known security vulnerabilities for various types of software. For this research it was decided to focus mainly on exploits that target the processes of the Linux kernel, as this is often the last vector to a full system compromise.

4.1 Test setup

The researchers have created an virtual machine that has sixteen different kernel options to boot from. Eight different version versions of the Linux kernel (x86) with and without Grsecurity enabled as shown in table 3. These versions have been carefully selected to cover the most amount of publicly available exploits, as discussed in section 4.2.

Each Grsecurity kernel has been compiled with the Debian kernel configuration and *High* Grsecurity profile, containing all the modules as shown in table 2. We have chosen to implement the highest level of Grsecurity to confirm the performance of all security features and that an exploit has possibly bypassed all of the security modules that Grsecurity offers. The different kernel version are required to cover vulnerabilities and proof-of-concept exploits only available in certain releases, while every kernel has a specific corresponding Grsecurity version.

Special attention for the configuration at build-time of the kernels that utilize the Grsecurity version 2.1.12 was required as some of the memory execution prevention features fail to be enabled by the configuration scripts. Only manual configuration of these items result in the *High* profile fully being enabled, although this exception only applied to version 2.1.12 in our testing.

The sources of the kernels used for testing can be retrieved from the repository. [2]

Linux kernel version	Grsecurity version	Kernel release
3.5.0	2.9.1	30 September, 2012
2.6.37.1	2.2.1	4 January, 2011
2.6.35.5	2.2.1	1 August, 2010
2.6.32.13	2.1.14	3 December, 2009
2.6.30.8	2.1.14	9 June, 2009
2.6.28.1	2.1.12	25 December, 2008
2.6.27.11	2.1.12	9 October, 2008
2.6.26	2.1.12	13 July, 2008

Table 3: Kernel & Grsecurity version

4.2 Kernel exploits

Based on the records of published CVEs [1] we filtered the vulnerabilities for Linux kernels from version *2.6.26* onwards with a corresponding malicious or proof-of-concept exploit. We gathered all available kernel exploits and found that the Exploit-db [7] repository is easily searched with the previously found CVE IDs.

The exploits that were verified to work on unprotected kernels are collected in table 4. Based on the requirements of the verified exploits we selected the best corresponding kernel versions to be hardened and tested, as previously discussed in section 4.1.

The kernel exploits collected all target a privilege escalation in some form or way, omitting exploits that merely try to cause a denial of service attack without resulting in a system compromise. We observed that most of the kernel exploits found can be grouped in a number of distinct exploit types. The most exploits use a form of memory corruption like a memory overflow [9] or NULL pointer dereference [8]. In a number of cases we found that an exploit misuses weak kernel processes in the form of process hijacking [6].

The sourcecode for all exploits can be found at both the Exploit-db and CVEdetails sources with the corresponding IDs found in table 4.

ExploitDB id	CVE #	Type	Exploit publish date
CVE: 2009-1186	EDB-ID: 8478	Memory overflow	2009-04-20
CVE: 2009-1337	EDB-ID: 8369	Process hijacking	2009-04-08
CVE: 2009-1897	EDB-ID: 9191	NULL pointer dereference	2009-07-17
CVE: 2009-2692	EDB-ID: 9435	NULL pointer dereference	2009-08-14
CVE: 2009-2695	EDB-ID: 9545	NULL pointer dereference	2009-08-31
CVE: 2009-3547	EDB-ID: 9844	NULL pointer dereference	2009-11-05
CVE: 2010-2959	EDB-ID: 14814	Memory overflow	2010-08-27
CVE: 2010-3437	EDB-ID: 15150	Memory overflow	2010-09-29
CVE: 2010-4258	EDB-ID: 15704	NULL pointer dereference	2010-12-07
CVE: 2013-1763	EDB-ID: 33336	Memory overflow	2013-02-24
CVE: 2013-1959	EDB-ID: 25450	Process hijacking	2013-05-14

Table 4: Exploits

5 Results

The exploits from table 4 were performed on the hardened kernels to determine if and how Grsecurity handles the prevention of the attack. In cases where the exploit was no longer successful but without a clear mention of the attempt in logfiles we made reasonable assumptions on which security modules were likely to have protected the specific vulnerability.

5.1 Blocked exploits

With the previously discussed exploits we set to test these on the potentially vulnerable system with Grsecurity installed. The following exploits were tested to be blocked by Grsecurity.

5.1.1 CVE: 2009-2695 - NULL pointer dereference

A problem with the handling of the *mmap_min_addr* setting by SELinux allowed for a NULL pointer dereference attack, placing malicious instructions in a low memory region by an application. This memory region is eventually executed by the kernel and enables root privilege escalation.
During our tests the first protection that triggered was the prevention of automatic kernel module loading. However, kernels patched with Grsecurity versions before 2.1.14 did allow for further exploiting because certain features were not enabled during kernel build-time. In particular the *KERNEXEC* security option, normally protecting against execution of injected instructions in kernel space, is of importance to this exploit.

The reason that earlier versions of Grsecurity fail to enable these security features is because of configuration conditions that are more strict in comparison to version from 2.1.14 onwards. We were unable to force kernel compilation with the *KERNEXEC* feature enabled, leaving us without clear results of this exploit on the 2.6.26, -27 and -28 kernels.

5.1.2 CVE: 2009-2692 - NULL pointer dereference

A vulnerability in the socket handling of kernels up to version 2.6.30 allows a NULL pointer dereference, similar to the exploit explained with CVE-2009-2695. Although the exploit succeeds on the kernels without *KERNEXEC* protection, Grsecurity 2.1.14 however does stop the execution of this exploit before the malicious instructions are executed by the kernel.

5.1.3 CVE: 2009-1897 - NULL pointer dereference

A vulnerability in the */dev/net/tun* device of the 2.6.30 and 2.6.31 kernels allows for exploiting with a NULL pointer dereference. Through the tun device the NULL address is set and for this particular implementation Pulseaudio is leveraged to further load the malicious instructions. Grsecurity's *PAX_NOEXEC* feature is able to prevent and detect the execution of the *NULL* address and therefore terminates the exploiting process.

5.1.4 CVE: 2010-4258 - NULL pointer dereference

A vulnerability allows this exploit to write directly to the kernel memory, although it requires another exploit to trigger this memory address. The exploit used for this trigger targets the Econet kernel module to execute the previous instructions from the kernel memory space. Grsecurity has a number of protections that prevent this exploit from being successful. The kernel symbols normally found in */proc/kallsyms* are not accessible, making it hard for the exploit to target a specific memory location. Predetermined memory locations wouldn't work as the kernel's address space is randomized through ASLR. It also blocks the automatic loading of kernel modules required for the Econet exploit with the *MODHARDEN* feature. Although not verified, we believe that the *NOEXEC* memory features of Grsecurity would prevent the execution of this memory region in case the memory regions would be known by the exploit and the Econet kernel module would already be loaded.

5.1.5 CVE: 2013-1763 - Memory overflow

Kernels up to version 3.7 had a vulnerability that allowed processes to receive root privileges through crafted arrays sent to the sock_diag function. The hardened kernel prevents this exploit because it restricts the kernel to not execute the crafted payload.

5.1.6 CVE: 2010-2959 - Memory overflow

The Controller Area Network (CAN) is targeted to perform an integer overflow attack, although it is likely that the CAN kernel module is yet to be loaded. The automatic loading of this module is blocked by the *MODHARDEN* feature, preventing this exploit unless this module was already loaded beforehand. Grsecurity would also stop the kernel from executing the prepared instructions from userspace in case the CAN module could've been exploited.

5.1.7 CVE: 2009-3547 - NULL pointer dereference

The *pipe* function in the kernel is exploited to trigger a NULL pointer dereference. The kernel tested was patched with Grsecurity 2.1.14, correctly enabling the *KERNEXEC* feature in our test-setup. The execution of this specific exploit is stopped because it highly depends on hard coded variables like memory locations that are harder to obtain or brute-force in a Grsecurity hardened situation thanks to ASLR. Further prevention would likely come from the memory protection features of Grsecurity that would detect execution of writable memory from a userspace memory region.

5.1.8 CVE: 2013-1959 - Process hijacking

This exploit gains higher privileges by manipulating the 'uid_map' and 'gid_map' files located in the */proc* filesystem. Grsecurity masks the information normally obtainable from */proc* from all processes but itself, rendering this exploit unsuccessful as the vulnerability is actually no longer present.

5.1.9 CVE: 2010-3437 - Memory overflow

A vulnerability in kernels before version 2.6.36 allowed for memory leaks, disclosing memory contents from chosen addresses. The exploit targets a kernel function and crafts the input for this function. Because the kernel symbols are shielded from userspace the exploit is prevented from directly targeting the weak function. Also, the memory randomization would result in wrong or no data to be returned, rendering this exploit unuseful.

5.2 Successful exploits

Grsecurity failed at preventing a number of exploits with the *High* security level. These exploits were looked at closely to determine if the vulnerability was out of scope or if different security modules would have been available with a custom Grsecurity configuration.

5.2.1 CVE: 2009-1337 - Process hijacking

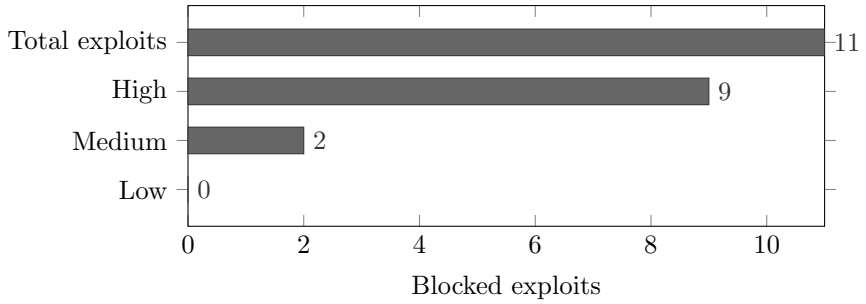
A shell binary is created by an unprivileged user which is modified to be owned by the root user with this exploit. The exploit sets ownership flags by letting a root parent process carefully execute these instructions in a predefined order. Grsecurity seems unable to detect the misuse of this bug in kernels up to version 2.6.29.

5.2.2 CVE: 2009-1186 - Memory overflow

This exploit abuses the *Udev* daemon, responsible for the handling of attached devices, and the communication channel with the kernel. The exploit instructs the Udev daemon from an unprivileged account to execute a previously prepared shell. As the Udev daemon is actually a kernel process this results in a successful privilege escalation. The exploit heavily relies upon the creation of a socket that communicates with the Udev daemon and a vulnerability in the daemon itself that allows for the execution of instructions of any unprivileged user by the kernel.
Grsecurity has no security measures to prevent the Udev daemon from executing these instructions. However, a possibility would be to restrict unprivileged users from setting up these sockets by using Grsecurity's custom 'GRKERNSEC_SOCKET' feature, although this would require more administration of policies.

5.3 Results by security level

By comparing the blocking module for each exploit with Grsecurity profiles we can correlate the effectiveness of each profile. We found that most of the exploits tested in section 5.1 were blocked by memory execution protections like *KERNEXEC* and *NOEXEC*, which are only found in hardened kernels configured with the *High* security level. From this we can derive that a lot of the features found in the *Medium* and *Low* security levels, as found in table 2, fail to protect against common kernel exploits as the memory execution would be the final step in performing an exploit.



6 Conclusion

The results show that Grsecurity offers an improvement in security. Nearly all exploits could be mitigated and a case can be made for those that were not. Grsecurity locked down the kernel however, it does not secure all processes that have root privileges on the same level. Installing Grsecurity increases the difficulty of a complete compromise and does provide a level of zero-day protection as claimed. Although unable to stop all attacks we feel that Grsecurity most certainly has a place in securing systems that run in an environment where there is unpriviliged access or potentially vulnerable software running with public access.

When using Grsecurity it is wise to take a close look into the build configuration. The installation menu makes it easier to setup Grsecurity but also abstracts away some of the details. This may lead to less modules installed than expected, like found with some of the kernel builds during our tests detailed in 4.1, lowering the security offered by Grsecurity.

7 Future work

During the research done into Grsecurity the following subjects and interesting avenues for future research arose. These subjects might be interesting to further explore:

7.1 Testing of userspace exploits

This research has focused solely on kernel exploits and has not taken applications vulnerabilities in consideration due to time constraints. Future research could be done on the effectiveness of published exploits on normal userspace software on Grsecurity hardened systems.

7.2 Revisit in time

The landscape of security and exploits is dynamic. The research that has been done has relied on software and exploits that are between 3 and 5 years old as of writing the paper. The landscape may change in the future making the results of this paper obsolete.

7.3 Usability review

The Grsecurity documentation and literature available warns that high security settings may result in an increasing amount of false positives and non-working software. We feel it would be worthwhile to see which applications are less forgiving in securely configured Grsecurity deployments and what the reason and workarounds would be for that.

References

[1] CVE Details. *CVEDetails.com - Security vulnerability datasource*. 2014.

[2] M. Berkelaar E. Bijnen. *Hardened kernel sources (Github repository)*. 2014.

[3] Michael Fox et al. *SELinux and grsecurity: A Case Study Comparing Linux Security Kernel Enhancements*.

[4] Grsecurity. *Grsecurity homepage*. 2014.

[5] Grsecurity.net. *Grsecurity Documentation*. 2014.

[6] SANS Institute. *Global Information Assurance Certification Paper*. 2003.

[7] Misc. *Exploit-db.com exploit repository*. 2014.

[8] Nelhage. *Much ado about NULL: Exploiting a kernel NULL dereference*. 2010.

[9] Owasp. *Buffer Overflow*. 2009.

[10] PaX Team. *PaX ASLR*. 2003.

Appendix

A Definitions

Kernelspace: Kernel space is made up of the memory regions strictly reserved for the Linux kernel, present at the end in the virtual memory space of every running process.

Userspace: Userspace refers to all memory regions of a typical application, strictly seperated from kernelspace.

Memory overflow: Grouping of overflow vulnerabilities where a memory buffer is written to outside of the expected boundaries. In some cases this may result in the overflown memory segments to be executed.

NULL pointer dereference: In rare cases it may be possible to trigger the execution of a NULL pointer, sometimes interpreted as the the very first memory region 0x00000000.

B Kernel repository

The sources of the hardened kernels along with the used configurations for the tests performed are available on Github at: `https://github.com/mikeberkelaar/grhardened`.

Linux kernel version	Source folder
3.5.0	Glinux-3.5
2.6.37.1	Glinux-2.6.37.1
2.6.35.5	Glinux-2.6.35.5
2.6.32.13	Glinux-2.6.32.13
2.6.30.8	Glinux-2.6.30.8
2.6.28.1	Glinux-2.6.28.1
2.6.27.11	Glinux-2.6.27.11
2.6.26	Glinux-2.6.26

Table 5: Kernel sources