Detection, Prevention, and Containment: A Study of grsecurity

Brad Spengler

http://www.grsecurity.net

spender@grsecurity.net

The Problem

The Problem

- Bugs in software cause unexpected results
- Unexpected functionality can result from errors in design, implementation, or configuration
- Bugs can often be wielded for malicious purposes by an attacker

Problems With the Current Solution

- Avoid / Identify / Fix
- Current state of security is a never ending rat race
- Endless cycle of vulnerability discovery and fixing

Problems With the Current Solution

- Ultimate goal of today’s security – removal of software bugs through auditing
- Security utopia – greatest result, though impossible to achieve

Problems With the Current Solution

- Auditing is expensive, slow, and requires a great deal of knowledge
- Auditing provides no guarantees about the security of the software
- Auditing cannot be fully automated
- EXAMPLE: format-string vulnerabilities

The (Attainable) Solution

The (Attainable) Solution

- Detection
- Prevention
- Containment

Advantages of the (Attainable) Solution

- Inexpensive
- Can be mostly automated
- Works for known and unknown bugs
- Allows administrators to focus more on administration (checking logs..etc) instead of rushing for the newest patch

Our solution:

grsecurity

Overview of grsecurity

Background on grsecurity

- Started in February 2001
- Initial release was for Linux 2.4.1
- Originally a port of Openwall to Linux 2.4

Goals of grsecurity

- Configuration-free operation
- Complete protection against all forms of address space modification bugs
- Feature-rich ACL and auditing systems
- Operation on multiple processor architectures and Operating Systems

Features of grsecurity

- A robust ACL system with an intelligent userspace administration tool
- Extensive auditing capabilities
- Measures to stop the most common methods of exploiting a system:
  - Address space modification
  - Races (specifically filesystem races, most common of which are /tmp races)
  - Breaking a chroot(2) jail

Features of grsecurity

- Supports sysctl so that it can be included with Linux distributions and allow the user to modify the options to his/her liking
- Netfilter module that drops connections to unserved TCP and UDP ports
- Many of the same randomness features as OpenBSD
- An enhanced implementation of Trusted Path Execution (TPE)

Detection in grsecurity

Detection in grsecurity

- Implemented in two forms
  - Auditing
  - Logging of real attacks
- Inode and device numbers used wherever possible
- Parent process info logged

Auditing

- Audited events include:
  - Exec (with arguments)
  - Chdir(2)
  - Mount(2)/unmount(2)
  - IPC (semaphore, message queue, shared memory) creation and deletion

Auditing

- Signals: SIGSEGV, SIGABRT, SIGBUS, SIGILL
- Failed forks
- Ptrace(2)
- Time changes (stime(2), settimeofday(2))
- Execs inside chroot(2)
- Denied capabilities

Prevention in grsecurity

Prevention in grsecurity

- Prevention is implemented through PaX and hardening certain sections of the kernel
- Hardened syscalls include:
  - Chroot(2)
  - Ptrace(2)
  - Mmap(2)
  - Link(2)/symlink(2)
  - Sysctl(2)

Prevention in grsecurity - PaX

- What is PaX?
  - PaX implements non-executable VM pages on architectures that do not support the non-executable bit (currently only ia-32, more to come)
  - PaX makes use of hardware-supported non-executable bits (still to be tested, but should work for alpha, parisc, and ia-64)
  - PaX provides full address space layout randomization (ASLR) for ELF binaries

Prevention in grsecurity - PaX

- How does PaX accomplish this?
  - Include/asm-<arch>/processor.h is modified to support executable and non-executable pages (if they don’t already exist)
  - Rest of kernel is modified to use the non-executable pages, applied to ELF and a.out binaries if they carry the required PaX flags (enabled by default)

Prevention in grsecurity - PaX

- Non-executable pages are made supervisor in the TLB; executable pages are left as user
  - If CPU is in user mode, access to the non-executable pages causes a page-fault which PaX handles
  - Makes up the core logic of how PaX works
  - Makes PaX ineffective against kernel overflows
- Mmap(2) and mprotect(2) restrictions/features
  - Disallows anonymous mappings with PROT_EXEC present – stops one method of arbitrary code execution (another method, mapping a file with PROT_EXEC, is handled by ACL system)

Prevention in grsecurity - PaX

- Causes mmaps (applies to libraries) to be mapped at random locations below 0x01000000 until it’s full, then above 0x40000000
  - Causes exploits to have to guess the library function address
  - Makes the address contain a NULL byte, which stops ASCII shellcode from calling a library function
- Keeps non-executable pages from being mprotected to executable
- No performance impact

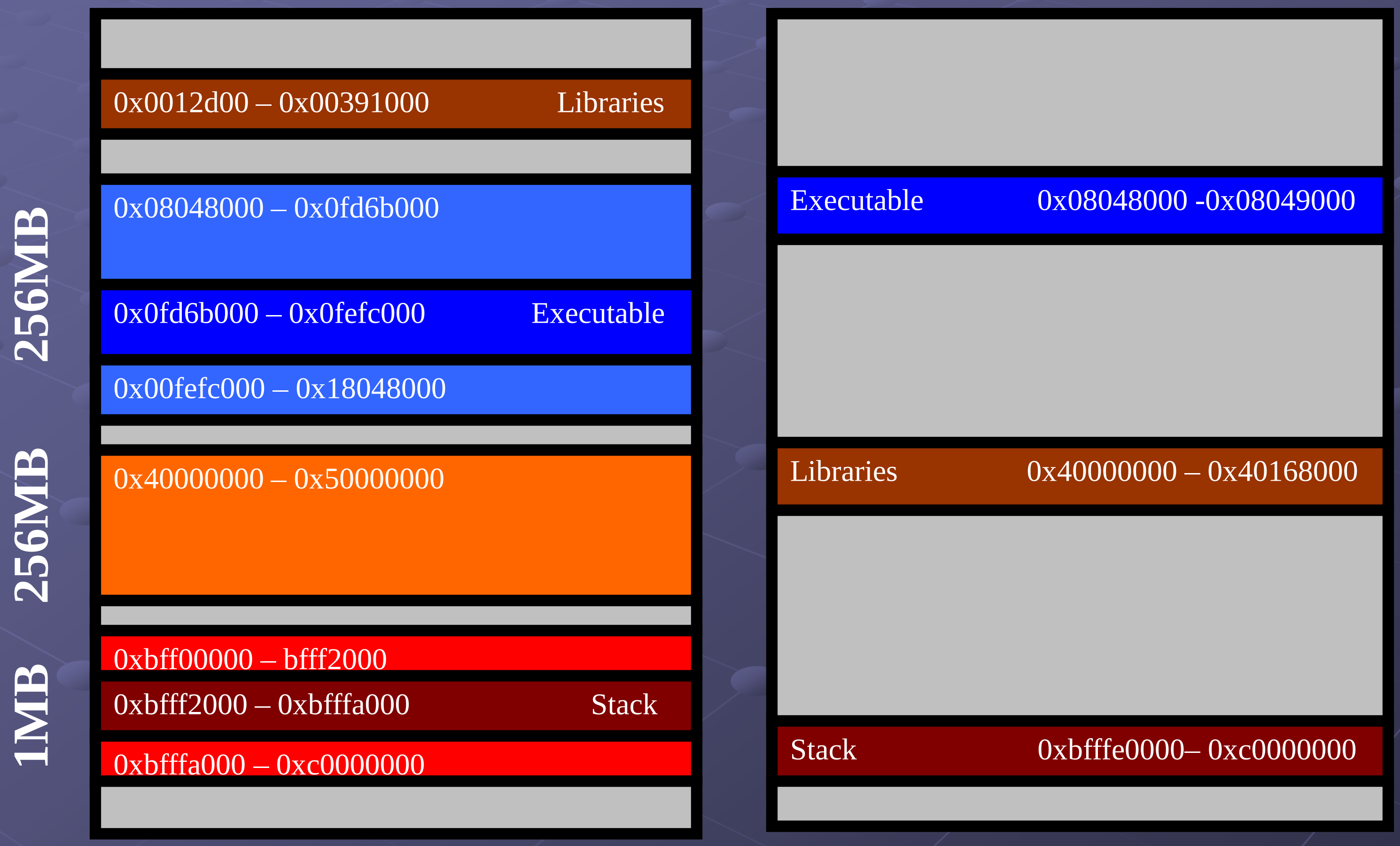
Prevention in grsecurity - PaX

- Full Address Space Layout Randomization (ASLR)
  - Randomizes the base of mmaps, stack, and executable (if the binary is ET_DYN)
  - Makes the leftover methods of exploitation a guessing game

	With no-exec	Without no-exec
Stack smashing	Impossible	Guess 16-bit
Heap overflow	Impossible	Guess 32-bit
Ret-to-libc	Guess 32 or 48-bit	Guess 32 or 48-bit

Prevention in grsecurity - PaX

PaX with Full ASLR | **Without PaX**



Prevention in grsecurity - PaX

- Full ASLR can only be bypassed in the case of information leak. While there’s nothing that can be done about software vulnerabilities that allow information leaking without crashing, we’ve implemented the following features to stop local users from obtaining information about the random base addresses:
  - Ptrace(2) restrictions in ACL system
  - Restricted /proc
- For 64-bit architectures, the randomness provided by full ASLR could be increased to 48/64/80 bits (the amount the attacker has to overcome is determined by the type of exploit)

Prevention in grsecurity - PaX

- What's in it for me?
  - No more arbitrary code execution
  - No more stack smashing, heap or bss overflow exploitation
  - No more return-to-libc exploitation
  - (Soon) no more arbitrary execution flow redirection

Prevention in grsecurity - PaX

- What's coming for this section of grsecurity?
  - New segmentation-based implementation of non-executable pages with an insignificant performance hit
  - Increased stack base address randomness to 24 bits
  - Binary instrumentation
    - Stops ret-to-libc by checkpointing execution flow changes
    - Ability to handle other vulnerabilities (eg. Stack based overflows, format string, info-leak)

Prevention in grsecurity

- OpenBSD randomness features
  - Random IP IDs
  - Random RPC XIDs
  - Random RPC privileged ports
  - Random PIDs

Prevention in grsecurity

- Random IP IDs
  - Uses Niels Provos' random IP ID generation function ported to Linux
    - Little entropy use
    - Values are not reused quickly
  - Useful for preventing OS fingerprinting and spoofed scans

Prevention in grsecurity

- Random RPC XIDs
  - Uses same random IP ID code
  - Useful for preventing RPC connection hijacking
- Random PIDs
  - Uses same random IP ID code
  - Properties of returned values make the function almost always return an unused PID even on heavily loaded servers

Prevention in grsecurity

- Prevents filesystem races since getpid() is sometimes used as part of a temporary filename
- Adds additional randomness to programs that use getpid(2) for srandom(3) seeding

Prevention in grsecurity

- Stealth netfilter module
  - Based on the fact that OS fingerprinting relies greatly on the packets sent in reply to those sent to unserved TCP or UDP ports
  - Matches unserved ports dynamically, so it can be used in shell-server environments
  - Slows down blocking port-scanners

Prevention in grsecurity

- Problems with chroot(2)
  - Easy to use it insecurely
  - Generally only filesystem-related functions care if a process is chrooted
  - Easy for a root user in chroot to break out

Prevention in grsecurity

- How we strengthen chroot(2):
  - Make syscalls unrelated to the filesystem chroot-aware
    - Deny double-chroots, pivot_root(2)
    - Restrict signals outside of chroot
    - Deny fchdir(2) outside of chroot
    - Deny mounting
  - Enforce chdir("/") upon chroot
  - Lower capabilities upon chroot

Containment in grsecurity

Containment in grsecurity

- Trusted Path Execution (TPE)
  - Keeps users from executing untrusted binaries (those not in root-owned non-world writable directories)
  - Hardened against evasion
    - Silent removal of glibc environment variables that allow arbitrary code execution (eg. LD_PRELOAD)
    - TPE checks implemented in mmap(2) (stops /lib/ld.so <executable> evasion)

Containment in grsecurity

- Grsecurity's ACL system
  - Process-based : Allowed for a large reduction in code base
  - ACL parsing handled via userspace, interacts with kernel via a /proc entry
    - Include directive
    - ACL analysis
      - $PATH
      - /etc/ld.so.conf
      - Auto-add libraries for ELF executables
      - /etc/lilo.conf

Containment in grsecurity

  - Uses LEX/YACC
  - Sends data to kernel in ready-to-use structures – further reduces necessary kernel code size
- Enable, disable, and administrator modes
- Hidden and protected processes
- Read, write, append and execute modes for file objects
- Inherit and hidden flags for file objects

Containment in grsecurity

- Capability support (including inheritance)
- Hardened against ACL evasion and privilege leaking
  - Ptrace restriction – user can only ptrace a process if the default ACL allows writing to it
  - Glibc environment variable handling
    - Performs correct handling, not just a denied exec if LD_ is found
    - Checks each path in glibc environment to see if the default ACL allows writing to it; if so, deny the exec and log pathname and environment variable used
  - Applies executable restrictions in mmap(2), not just execve

Containment in grsecurity

- Human readable configuration files
- Insignificant performance impact due to efficient searching algorithms (hash tables == O(1))

Containment in grsecurity

- What’s coming for the ACL system?
  - Redesign to become more modular and allow quicker implementation of new features
  - Intelligent learning mode resulting in a least-privilege system with little or no configuration necessary
  - Support of fine-grained resource restrictions and something similar to nergal’s segvguard
  - Time-based ACLs
  - Merging of GID-based grsecurity features
  - Role-Based Access Control (RBAC)

Containment in grsecurity

- Domain-based authentication support

Performance

Performance of ACL system

- Completed 150 runs of 16 dbench processes
- Average throughput with ACL system was larger than a clean kernel
- Standard deviation was 5MB/s, which was larger than the difference of throughput
- RESULT: The ACL system causes no noticeable performance hit on filesystem access

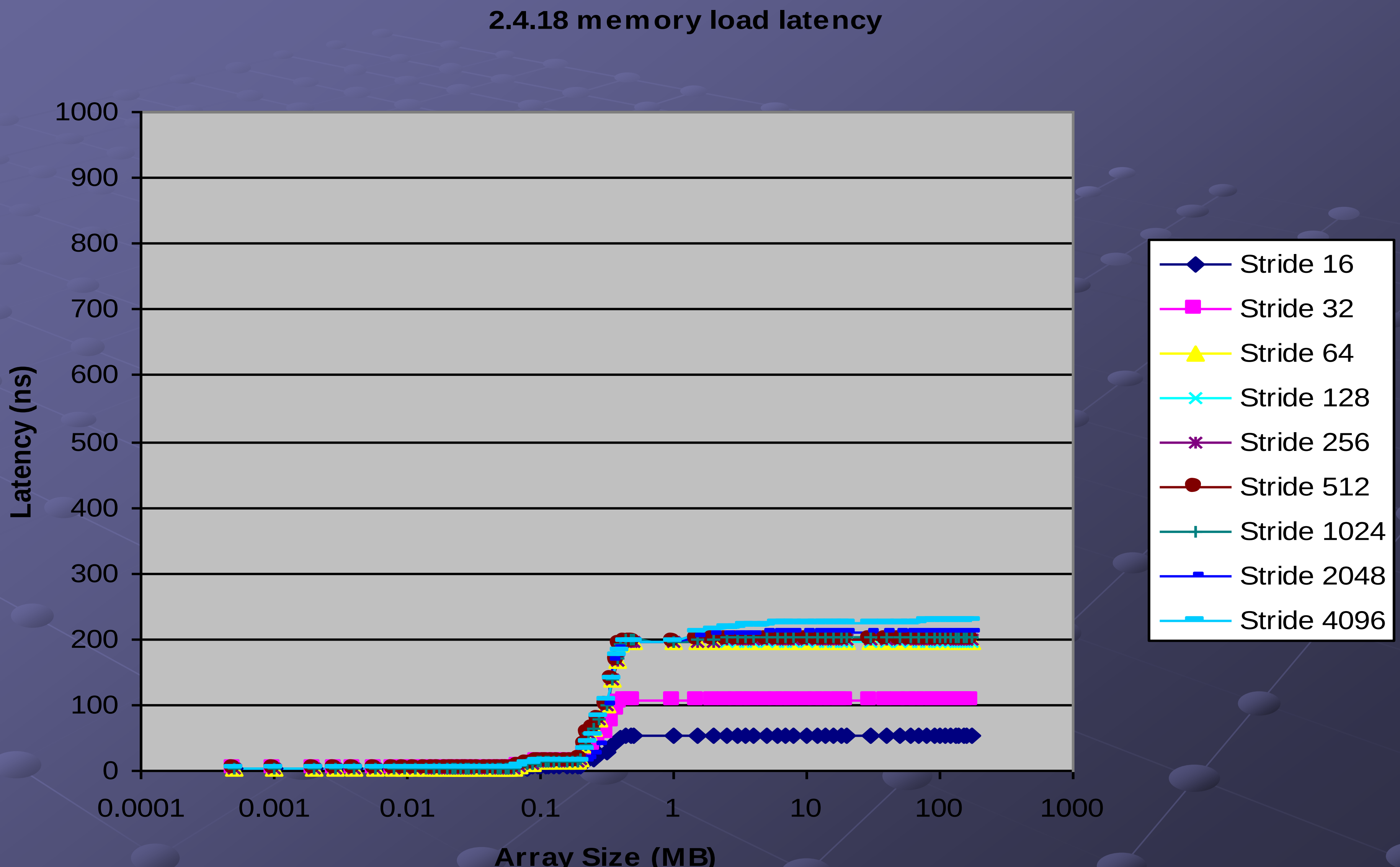
Performance of ACL system

- Results of kernel compile benchmark:
  - Total time with ACL system – 265.86 seconds
  - Total time w/o ACL system – 264.94 seconds
  - .4% performance hit
- Performance hit only due to execs in compiling and making – search is called twice, acl label is copied, acl label is set, checks are performed on the environment

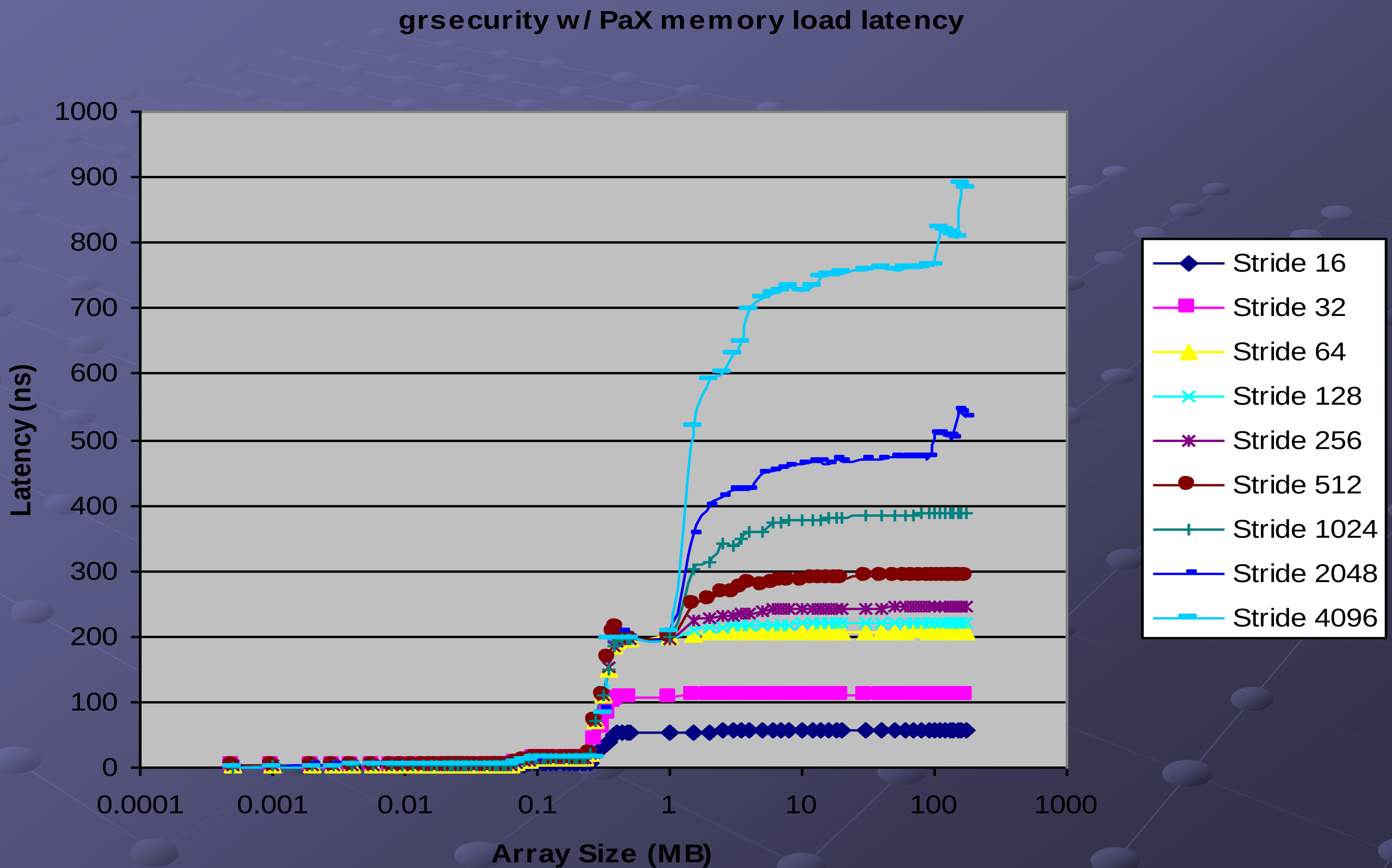
Performance with PaX

- Memory load latency microbenchmarks
- MySQL benchmarks (real life example)
- Test system:
  - Dual AMD XP 1600+
  - 512MB PC2100 ECC DDR registered RAM
  - 266mhz FSB
  - 80GB ATA100 5400RPM HD

Performance with PaX



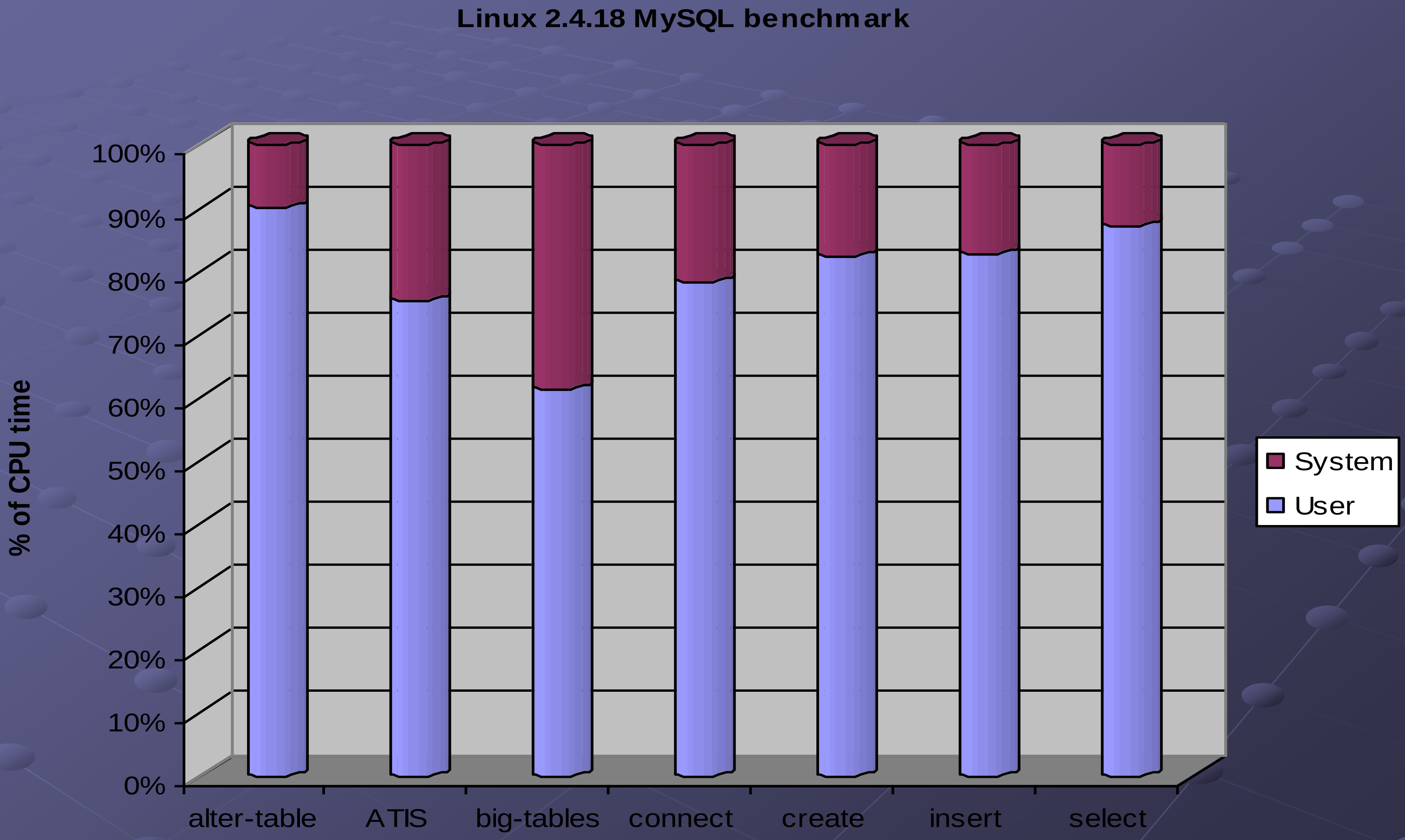
Performance with PaX



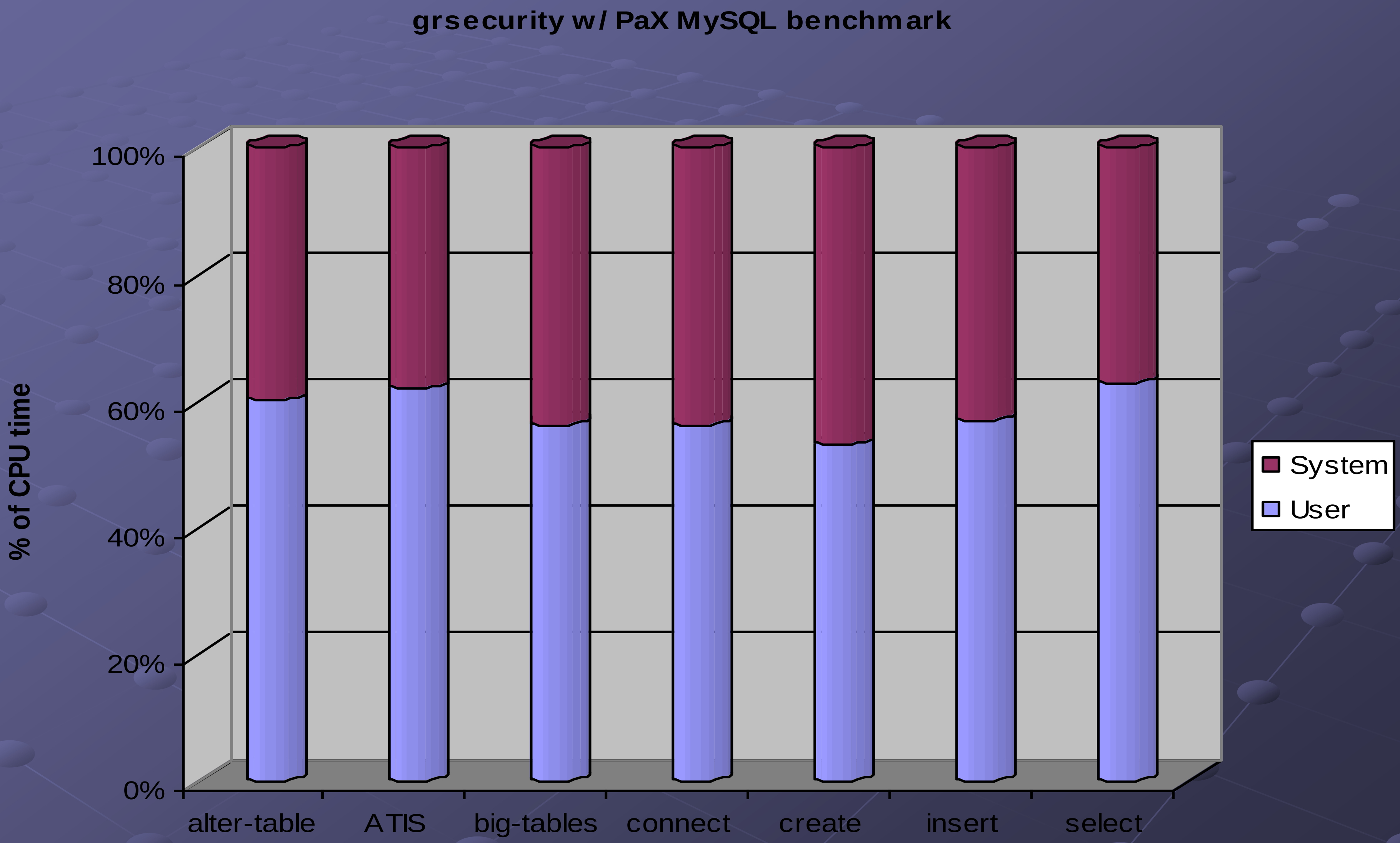
Performance with PaX

- Athlons encounter less performance hit partially due to their 256 entry DTLB (4KB page x 256 = 1MB)
- PaX starts showing its performance impact when the DTLB is full and expired entries are replaced
- Performance with PaX can only be determined by the size and type of memory accesses performed by an application

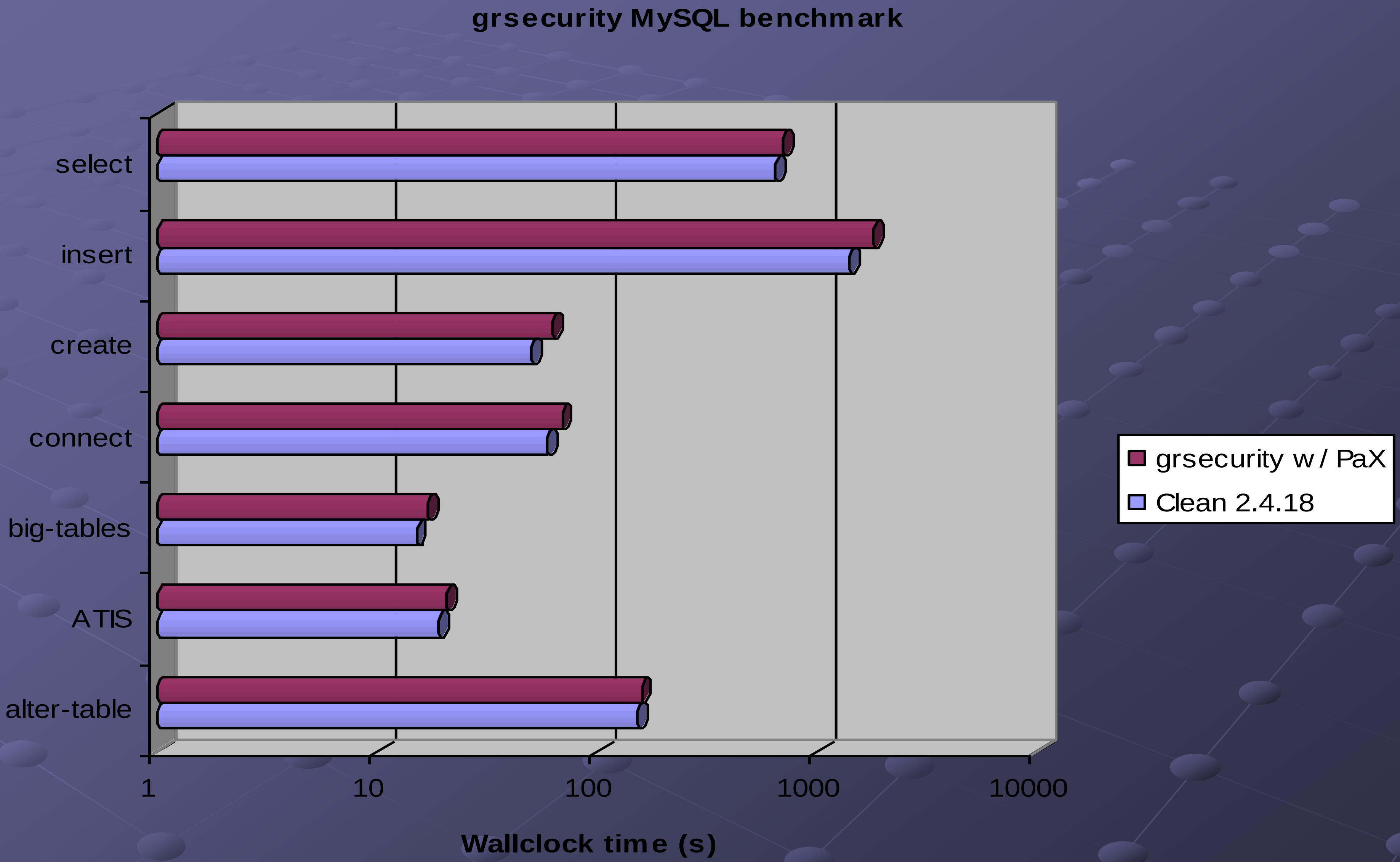
Performance with PaX



Performance with PaX



Performance with PaX



Performance with PaX

- A result weighted according to an actual system’s load shows that for MySQL, PaX caused an overall performance hit of 13%
- Since the memory access patterns of each test were different, the performance hits for each test ranged from 3% - 20%

For More Information…

- grsecurity's ACL documentation: http://www.grsecurity.net/gracldoc.htm
- PaX

  http://pageexec.virtualave.net

THANKS

- PaX Team
- Tim Yardley
- Michael Dalton - grsecurity