PaX (http://pageexec.virtualave.net)

The Guaranteed End of Arbitrary Code Execution

Who am I?

- Brad Spengler
  - The only grsecurity developer
  - NOT a PaX developer
  - Computer Engineering major, Mathematics minor

What is PaX?

- Quite simply: the greatest advance in system security in over a decade that you've never heard of
- Less simply: It provides non-executable memory pages and full address space layout randomization (ASLR) for a wide variety of architectures.

Outline

- PaX “lecture”
  - SEGMEXEC
  - PAGEEXEC
  - KERNEXEC
  - ASLR
    - RANDMMAP
    - RANDEXEC
    - ET_DYN
    - RANDKSTACK

Outline (cont.)

- How grsecurity is involved in PaX's strategy
- Factual comparison with OpenBSD's W^X
  - ASLR comparison
  - Any guarantees?
  - The "subtle concept" of mprotect
- Factual comparison with Exec Shield
  - ASLR comparison
  - Any guarantees?
  - Mprotect

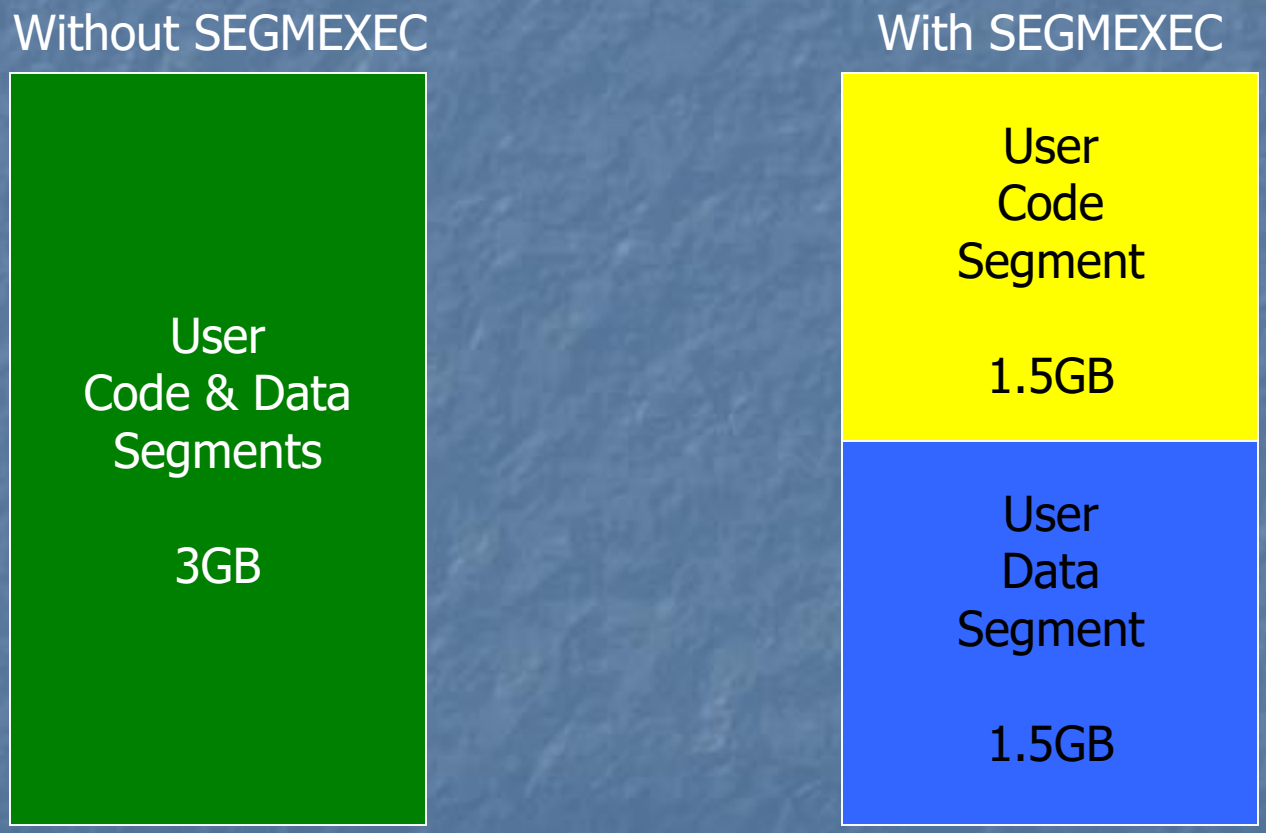
PaX - SEGMEXEC

- SEGMEXEC is PaX's implementation of per-page non-executable user pages using the segmentation logic of IA-32 (Intel x86 architecture) and virtual memory area mirroring (developed by PaX).

PaX – SEGMEXEC (cont.)

- The segmentation logic is fairly straightforward:
  - Data Segment (DS)
  - Code Segment (CS)
- There exist these two segments for user pages as well as kernel pages.
- PaX splits the address space down the middle: the bottom half for data, the top for code.
- Segmentation is a “window” into the address space
- No performance hit

PaX – SEGMEXEC (cont.)



PaX – SEGMEXEC (cont.)

- PaX's VMA mirroring involves duplicating every executable page in the lower half of the address space into the upper half.
- Instruction fetch attempts at addresses located in the data segment that don't have any code located at its mirrored address will cause a page fault. PaX handles this page fault and kills the task.

PaX – SEGMEXEC (cont.)

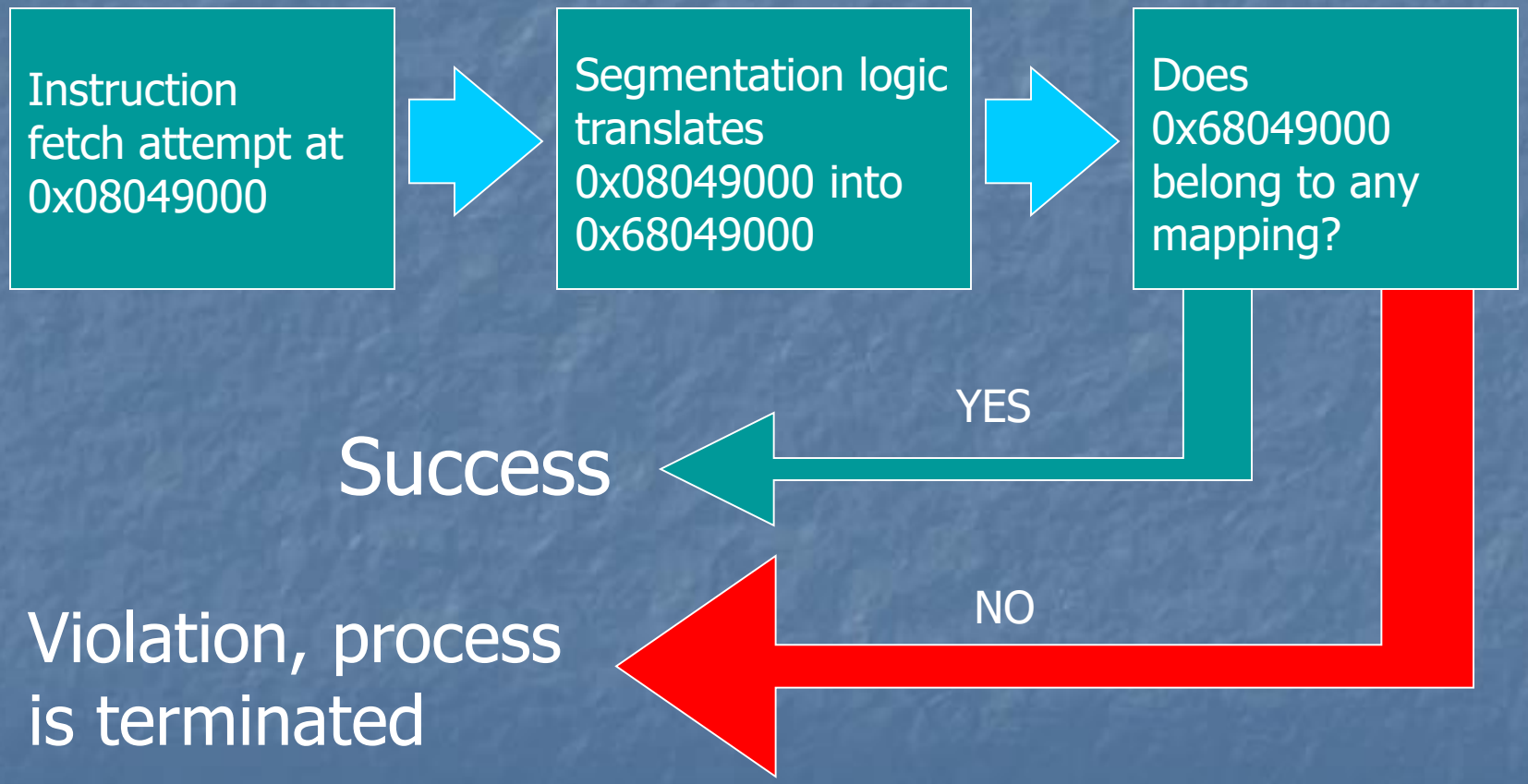
```
08048000-0804c000 r-xp /home/spender/cat
0804c000-0804d000 rw-p /home/spender/cat
0804d000-08073000 rw-p
40000000-40014000 r-xp /lib/ld-2.3.2.so
40014000-40015000 rw-p /lib/ld-2.3.2.so
40015000-40016000 rw-p
4001e000-40145000 r-xp /lib/libc-2.3.2.so
40145000-4014a000 rw-p /lib/libc-2.3.2.so
4014a000-4014c000 rw-p
4014c000-402d1000 r--p /usr/lib/locale/locale-archive
bfffe000-c0000000 rw-p
```

Without SEGMEXEC

```
08048000-0804c000 r-xp /home/spender/cat
0804c000-0804d000 rw-p /home/spender/cat
0804d000-08079000 rw-p
20000000-20014000 r-xp /lib/ld-2.3.2.so
20014000-20015000 rw-p /lib/ld-2.3.2.so
20015000-20016000 rw-p
2001e000-20145000 r-xp /lib/libc-2.3.2.so
20145000-2014a000 rw-p /lib/libc-2.3.2.so
2014a000-2014c000 rw-p
2014c000-202d1000 r--p /usr/lib/locale/locale-archive
5ffff000-60000000 rw-p
68048000-6804c000 r-xp /home/spender/cat
80000000-80014000 r-xp /lib/ld-2.3.2.so
8001e000-80145000 r-xp /lib/libc-2.3.2.so
```

With SEGMEXEC

PaX – SEGMEXEC (cont.)



PaX - PAGEEXEC

- PAGEEXEC was PaX's first implementation of non-executable pages.
- Because of SEGMEXEC, it's not used anymore on x86 (so I won't discuss the implementation).
- Platforms which support the executable bit in hardware are implemented under PAGEEXEC (currently alpha, ppc, parisc, sparc, sparc64, amd64, and ia64)

PaX - KERNEXEC

- KERNEXEC is PaX's implementation of proper page protection in the kernel
  - 'const' finally means read only in the kernel
  - Read-only system call table
  - Read-only interrupt descriptor table (IDT)
  - Read-only global descriptor table (GDT)
  - Data is non-executable
  - Uses the same concept of segmentation as SEGMEXEC
  - Cannot co-exist with module support (currently)

PaX - ASLR

- Full ASLR randomizes the locations of the following memory objects:
  - Executable image
  - Brk-managed heap
  - Library images
  - Mmap-managed heap
  - User space stack
  - Kernel space stack

PaX – ASLR (cont.)

- Notes on amount of randomization:
  - The following values are for 32bit architectures. They are larger on 64bit architectures, though not twice as large (since they generally don't use 64 bits for the address space).
    - Stack – 24 bits (28 bits for argument/environment pages)
    - Mmap – 16 bits
    - Executable – 16 bits
    - Heap – 12 bits (or 24 bits if executable is randomized also)

PaX – ASLR (cont.)

- The randomizations applied to each memory region are independent of each other
  - Because PaX guarantees no arbitrary code execution, exploits will most likely need to access different memory regions.
  - So, if the exploit needs access to libraries and the stack, the bits that must be guessed are the sum of the two regions: 40 bits (or 44). The chance of such an attack succeeding while depending on hard coded addresses is effectively zero.

PaX – ASLR (cont.)

```
08048000-0804c000 r-xp /home/spender/cat
0804c000-0804d000 rw-p /home/spender/cat
0804d000-08078000 rw-p
4edaa000-4edbe000 r-xp /lib/ld-2.3.2.so
4edbe000-4edbf000 rw-p /lib/ld-2.3.2.so
4edbf000-4edc0000 rw-p
4edc8000-4eeef000 r-xp /lib/libc-2.3.2.so
4eeef000-4eef4000 rw-p /lib/libc-2.3.2.so
4eef4000-4eef6000 rw-p
4eef6000-4f07b000 r--p /usr/lib/locale/locale-archive
bf3dc000-bf3dd000 rw-p
```

```
08048000-0804c000 r-xp /home/spender/cat
0804c000-0804d000 rw-p /home/spender/cat
0804d000-08070000 rw-p
43d8c000-43da0000 r-xp /lib/ld-2.3.2.so
43da0000-43da1000 rw-p /lib/ld-2.3.2.so
43da1000-43da2000 rw-p
43daa000-43ed1000 r-xp /lib/libc-2.3.2.so
43ed1000-43ed6000 rw-p /lib/libc-2.3.2.so
43ed6000-43ed8000 rw-p
43ed8000-4405d000 r--p /usr/lib/locale/locale-
archive
b54f9000-b54fa000 rw-p
```

Two runs of a binary with stack, mmap, and heap randomization

PaX – ASLR (cont.)

- RANDKSTACK
  - Randomizes the kernel's stack
  - Randomized on each system call, so info-leaking the randomization is useless
  - Randomizes 5 bits of the stack. Brute forcing generally shouldn't be possible, as each attempt will most likely crash the kernel.

PaX – ASLR (cont.)

- ET_DYN
  - Special type of ELF binary (the same used for shared libraries)
  - Position independent code (PIC)
  - Allows for relocation of the binary at a random location
  - Needed to achieve Full ASLR
  - Requires a recompile and re-link of applications
  - Adamantix and Hardened Gentoo have adopted these changes.

PaX – ASLR (cont.)

- RANDEXEC
  - Randomizes the placement of code in ET_EXEC binaries.
  - Uses the same segmentation feature as SEGMEXEC.
  - Code in an ET_EXEC binary is mirrored at a random location. The code still exists as data in the data segment.
  - When execution of the program enters the binary image, a page fault is raised and analyzed.
  - The analysis checks to see if the entry into the binary image was legitimate or caused by a ret-to-libc style attack. If it was legitimate, execution is redirected into the randomized mirror; otherwise, the application is killed.
  - RANDEXEC can cause false positives in certain applications. Also since it does not randomize data in the binary, it is not a replacement for ET_DYN. RANDEXEC was developed merely as a proof of concept.

How grsecurity is involved in PaX's strategy

- To truly achieve the guarantee of no execution of arbitrary code, grsecurity must be used. The ACL/RBAC system or TPE can be used to ensure that an attacker can't create a file with his payload in it, and mmap that executable via a ret-to-libc attack on the process.
- Protection against brute-forcing attacks is also part of PaX's strategy. This is handled within grsecurity's ACL/RBAC system by either denying execution of the app for a single user or for everyone (depending on whether the process was a network daemon or not).

Factual Comparison of PaX and W^X

- PaX
  - Guaranteed no execution of arbitrary code
  - 24/28 bit stack randomization
  - 16 bit mmap randomization
  - Completely implemented in the kernel. Can be implemented transparently and retain binary compatibility with all distributions.
- W^X
  - No guarantees about arbitrary code execution
  - 14 bit stack randomization
  - 16 bit mmap randomization
  - Required a complete recompilation/re-linking of user space. Broke binary compatibility with all previous OpenBSD releases.

Factual Comparison of PaX and W^X (cont.)

- PaX
  - Cuts usable address space in half (though this can be changed if it becomes a problem)
  - Two methods for randomizing the executable base (though ET_DYN is the correct method)
  - Support for non-executable and read-only kernel pages on i386
- W^X
  - More usable address space, but fragmented
  - As of the latest release, no method for randomizing the executable base
  - No support for non-executable or read-only kernel pages on i386

Factual Comparison of PaX and W^X (cont.)

- PaX
  - Per-system call kernel stack randomization
  - Brk-managed heap randomization
  - Ability to enable/disable all features on a per binary basis
  - No read-only GOT/PLT/.ctors/.dtors (yet)
- W^X
  - No kernel stack randomization
  - No brk-managed heap randomization
  - No method of toggling features on a per binary basis
  - Read-only GOT/PLT/.ctors/.dtors

Factual Comparison of PaX and W^X (cont.)

- PaX
  - Supports the same user space features on i386, alpha, ppc, parisc, sparc, sparc64, amd64, and ia64.
  - Supports a per-page implementation of non-executable pages on ppc
- W^X
  - Supports the same user space features on i386, alpha, ppc, parisc, sparc, and sparc64. (giving benefit of the doubt here as some work is yet to be done on ppc, possibly others)
  - Supports a segmentation-based implementation of non-executable pages on ppc that cannot guarantee W^X on large memory loads.

Functional Comparison of PaX and W^X

- As noted, there are many differences between PaX and W^X, but what do these technical differences mean in terms of effectiveness against real-life exploit scenarios?
  - W^X will not prevent exploitation of the kernel
  - The .bss and heap can be used in exploits to store data for the payload at a known location on OpenBSD
  - OpenBSD's mmap randomization is somewhat useless at preventing ret-to-libc style attacks since the PLT in the executable image is not at a randomized location and will allow for a similar attack.
  - On OpenBSD, attackers are not limited to the code that resides in a task to complete their exploit.

Rebuttal of arguments for W^X

- Claim:
  - "randomizing load order indirectly leads to random addresses. sshd loads 8 libraries. 8! is 40000, meaning if you have some return to libc attack, libc could be at one of many many different locations.

    in short:

    attack type: return to libc.

    solution: move libc."

  tedu@openbsd.org : http://www.deadly.org/article.php3?sid=20031009110855&mode=flat

Rebuttal of arguments for W^X (cont.)

- Rebuttal:
  - The statement that random load order of 8 libraries results in 40,000 possible orders does not have a direct relation to security. The assumption is made that for a successful attack, one would need data/code from each of the 8 libraries, when in reality, only one is needed. So in the presence of only random load order, focusing the attack on the first library will give you a 1 in 8 chance of success. This is hardly anything that can be called security.

Rebuttal of arguments for W^X (cont.)

- Claim: OpenBSD cannot protect against attacks using mprotect because it would violate POSIX, and OpenBSD does not violate POSIX.
  - > > We don't break anything that standards or defacto standards require. (Theo de Raadt)

    > You do break POSIX as pointed out above. (PaX Team)

    False. Now go away. (Theo de Raadt)

    http://groups.google.com/groups?selm=200304171509.h3HF9N5t023465%40cvs.openbsd.org.lucky.openbsd.misc&oe=UTF-8&output=gplain

Rebuttal of arguments for W^X (cont.)

- Rebuttal:
  - OpenBSD violates POSIX
    - "Indeed. None of the *BSD systems currently checks for PROT_EXEC in this case."

      miod@openbsd.org agreeing to POSIX violation in mmap()

      http://www.deadly.org/article.php3?sid=20031009110855&mode=flat
  - OpenBSD's POSIX compliance has not been verified formally or informally by any third party. Thus their claims of compliance are opinions and not fact.

Rebuttal of arguments for W^X (cont.)

- PaX does not violate POSIX by restricting mprotect()
  - "If an implementation cannot support the combination of access types specified by *prot*, the call to *mprotect*() shall fail."

    http://www.opengroup.org/onlinepubs/007904975/functions/mprotect.html

Rebuttal of arguments for W^X (cont.)

- Claim: PaX "goes too far" and breaks applications. W^X does not break anything.
  - > That's when you modify non-compliant software to bring it in line

    > with what the standard says. (PaX Team)

    False. You go too far. (Theo de Raadt)

    "Our W^X changes break nothing." (Theo de Raadt)

    http://groups.google.com/groups?selm=200304171509.h3HF9N5t023465%40cvs.openbsd.org.lucky.openbsd.misc&oe=UTF-8&output=gplain

    http://groups.google.com/groups?selm=200304170012.h3H0C45t025999%40cvs.openbsd.org.lucky.openbsd.misc&oe=UTF-8&output=gplain

Rebuttal of arguments for W^X (cont.)

- Rebuttal:
  - OpenBSD breaks binary compatibility, PaX does not.
    - Binaries for OpenBSD that were incorrect to begin with (such as assuming malloc() returns executable memory) will be broken under W^X and have no way to be corrected, since they do not support per-binary disabling of features. In PaX, a single command can correct the problem. This is a secure-by-default design.
  - The upstream release of XFree86 < 4.3 assumed malloc() returns executable memory for its module loader. Some point to this as PaX breaking XFree86, when it was a bug in XFree86 that simply was not important before. There are other similar bugs involving libGL and various drivers. Redhat fixed these bugs in XFree86 when Exec Shield was developed.

Additional comments on PaX and W^X

- OpenBSD has yet to release any sort of formal documentation on the design and implementation of W^X. PaX's has been available at http://pageexec.virtualave.net/docs/
- There has been no public discussion of any kind of attack model for W^X, while PaX's is very well defined. PaX was developed to defeat entire classes of exploits. W^X provides no guarantees and seemingly attempts only at picking away at several kinds of bugs (such as linear stack overflows) through the use of many assorted features. As shown earlier, the merits of some of these features are debatable, and it makes it increasingly difficult for OpenBSD to ever have any kind of toggling feature. Such a toggling feature is important not only to keep binary compatibility but also for easier debugging.

Factual comparison of PaX and Exec Shield

- For the most part, Exec Shield and W^X are similar (in that they both provide a subset of the features of PaX), so I will not give a point-by-point analysis. However, some differences between PaX and Exec Shield are:
  - Exec Shield uses less randomization than PaX in every region, though it randomizes the same areas.
  - To randomize the executable image, Exec Shield makes use of Redhat's PIE (Position Independent Executable).
  - Exec Shield cannot even guarantee that when a task is fully loaded in memory, that there do not exist memory regions that are both writable and executable, even if an application did not request such mappings.
  - Exec Shield recently discovered a bug (an off-by-one page), due to someone running paxtest on an Exec Shield machine, that resulted in a page of memory being writable and executable that was assumed otherwise. This bug was present ever since the first release of Exec Shield.
  - Exec Shield does nothing against kernel exploitation

References

- OpenBSD http://www.openbsd.org
- PaX http://pageexec.virtualave.net
- Exec Shield http://people.redhat.com/mingo/exec-shield/
- Hardened Gentoo http://www.gentoo.org/proj/en/hardened/
- Adamantix http://www.adamantix.org

Questions?

- Thanks for attending and your interest in PaX. With whatever time remaining, I'd be glad to answer any questions about PaX or grsecurity.