Converting OpenBSD to PIE

Pascal Stumpf

ABSTRACT

*Position-independent executables (PIEs) are the last step on the journey to a fully randomised address space on OpenBSD, with the goal of providing improved defense against return-oriented programming. This paper details the measures undertaken to successfully make this conversion on a broad, system-wide scale. It also provides a perspective on both the future of practically deployed ROP mitigations and the prevalence of such features (including PIE) on other operating systems, such as *BSD, Linux and Windows.*

Return-oriented programming

With the advent of data execution prevention techniques in most common operating systems,[1] exploit writers have turned to return-oriented programming (ROP) to circumvent these measures.[2] Instead of relying on the ability to simply execute uploaded shellcode, ROP utilises existing code present in either shared libraries or the executable itself. Diverting the main program's control flow, the attacker overwrites the return address with an address inside the program or a library, executing the code in the specified location until a return instruction is encountered, jumping to the next attacker-controlled location.[3] These chunks of code are commonly called 'gadgets'.

By chaining together these gadgets, the attacker will want to execute a meaningful piece of code, e.g. execute /bin/sh or open a network socket. Thus, it is desirable to, given a sufficiently large and commonly used body of code (such as libc), collect gadgets that allow for Turing-complete functionality. While the attacker is restricted to intentional return-and-restore instructions on architectures with a fixed instruction length (such as sparc64),[4] x86 allows for interpreting any sequence of 0xc3 as a ret instruction.[5] Furthermore, the use of ret has been demonstrated to not be required.[6] Using this knowledge, the process of 'gadget-mining' consists of searching for (intended and unintended) ret or jmp opcodes and trying to find a usable preceding instruction sequence. Automated tools are available for this task.[7]

Countermeasures: ASLR

It is obvious that ROP is threat to be taken seriously by both application writers[8] and operating system vendors. One commonly deployed defense mechanism is address space layout randomisation (ASLR). By randomising the location of code

1. See for instance on details of the OpenBSD W^X implementation: http://www.openbsd.org/papers/ven05-deraadt/index.html
2. http://seclists.org/bugtraq/1997/Aug/63
3. First generalised from using whole functions by Shacham (2007).
4. Buchanan et al. (2008), p. 29.
5. Roemer et al. (2012), p. 6f.
6. Checkoway et al. (2010).
7. https://github.com/JonathanSalwan/ROPgadget
8. Most recently, see src/lib/libssl/src/crypto/x86cpuid.pl, Revision 1.4 in the OpenBSD source tree. The OPENSSL_indirect_call function is still present in upstream OpenSSL as of 27. Nov. 2014.

segments on subsequent binary invocations, it is no longer possible for an attacker to easily predict the location of the desired gadgets. OpenBSD has shipped with random library ordering and random library base addresses for more than a decade.[9]

Introducing PIE

But while these mitigations counteract any attack that tries to make use of library code, the executable itself remained at a fixed position for a long time: The PIC model that allowed randomisation of shared libraries was not available for the executable. This changed when support for the -fpie and -fPIE flags was added to the GNU Compiler Collection,[10] and the -pie flag to GNU Binutils,[11] respectively, and the project to support position-independent executables in OpenBSD was taken up by Kurt Miller.[12] Subsequently, support for compiling, linking, running and debugging PIE executables was added to the system and included in the 4.5 release on almost all platforms.

Effort to deploy PIE on a system-wide basis with very few exceptions was also started. It was taken up again at the g2k12 hackathon, where an initial assessment of the impact on third-party applications (the OpenBSD ports tree) as well as the OpenBSD build system was made. With ‘security by default’ being one of the project’s main goals, the approach taken was naturally to alter the toolchain to generate position-independent executables by default without the need to pass any further flags to the compiler invocation.

The implementation in OpenBSD’s copy of GCC is straightforward: Depending on the architecture, the build system passes the default value of the flag_pie variable (1 for the small -fpie, 2 for the big -fPIE model, required on sparc64, powerpc and alpha), with profiling code (-p, -pg) being the only exception. The linker, too, was converted to generate PIE executables by default on PIE architectures. This also made it necessary to introduce a new option, called -nopie, to turn off PIE if needed. But soon it was discovered that this new default had unexpected side effects: Given the command line ‘cc -fno-pie -static foo.c -o foo’, ‘foo’ would end up being dependent on the runtime linker ld.so. While these are technically the correct semantics (-static meaning ‘link libraries statically’, not ‘generate a static executable’), it was decided that making -static imply -nopie would be less astonishing to the average user.

PIE is switched on globally depending on architecture.[13] The 5.6 release has shipped with the following architectures using PIE by default: alpha amd64 hppa i386 mips64 mips64el powerpc sh sparc64, which comprise the majority of those supported by OpenBSD.

Non-PIE components of the base system

Relocation of binaries is dependent on the runtime linker. Therefore, there are several code segments in the system which cannot be relocated this way as they must continue to function independently. This includes the early bootstrap code, the kernel, and, before the 5.7 release, static binaries. For the former two, it was simply a matter of adding the correct flags to the Makefiles (CFLAGS+=-fno-pie, LDFLAGS+=-nopie), while in order to mark specific userland binaries as non-PIE, a switch called NOPIE (analogous to NOPIC) was added, which automagically sets the correct flags.

As of the upcoming 5.7 release, the only userland program using NOPIE is gcc. The reason for this is its reliance on sbrk(2) in order to map C++ precompiled headers to a fixed location in memory. OpenBSD’s mmap(2) deliberately makes no such guarantee, and with the base address of the main program being randomised the brk space will move around as well. With no proper serialisation, and absolute pointers encoded in the on-disk PCH file,[14] it was a choice between losing PCH support and marking gcc as non-PIE. Given the large performance impact of PCHs and the relatively small attack surface of a compiler, PIE was disabled for gcc.

PIE code is typically bigger than standard (relocatable) code; therefore, it is only logical to also turn it off in a space-constrained environment like the OpenBSD installer ramdisk. In order to achieve this, it was not sufficient to pass the usual compiler and linker flags to the build system, because the system libraries (like libc.a) linked into the final crunchgen(8) executable still contained the bigger PIE code. Rather, only the objects that are actually required by the final binary (determined using a link map, ld -M) are now rebuilt with -fno-pie.

9. http://www.openbsd.org/34.html
10. https://gcc.gnu.org/ml/gcc-patches/2003-06/msg00140.html
11. https://sourceware.org/ml/binutils/2003-05/msg00832.html
12. http://www.openbsd.org/papers/nycbsdcon08-pie/
13. See src/share/mk/bsd.own.mk.
14. See src/gnu/gcc/libcpp/pch.c.

PIE has to be turned off for profiling (-p, -pg); this is due to the fact that the gprof(1) tool heavily relies on a fixed load address for the executable in order to generate a call graph from the profile data (gmon.out). In order to deal with position-independent code, both in PIEs and shared libraries, the profiling stack would require a major overhaul.

Issues in the ports tree

Fallout in the OpenBSD ports tree, consisting of over 7500 packages as of the 5.3 release (the first using PIE by default) and around 9000 as of today, was comparatively small. The first class of ports that needed modifications were compilers: Since the default behaviour of the linker changed to try linking with -pie per default, ports compilers needed to adopt the default-PIE model as well. Among those are the various copies of gcc, implementing an arch-based switch similar to that of the base system, as well as LLVM:

• devel/llvm

• lang/gcc/*

• lang/gfortran

• lang/g77

Other compilers that lack the ability to generate PIE code, but depend on the system linker /usr/bin/ld need to pass -nopie to the linker per default, such as:

• lang/fpc

• lang/ghc

• lang/gprolog

• lang/sbcl

The OpenBSD ports tree also includes bootloaders and the like, requiring the same treatment as in the base system:

• sysutils/grub

• sysutils/memtest86+

Some applications insist on using optimised inline assembler instructions. Sometimes, this code does not account for the requirements imposed by PIE (or PIC, for that matter): This is especially a concern on the notoriously register-starved i386 architecture. For instance, the i386 'PIC register' %ebx must not be clobbered (or saved and restored). Also, the additional register pressure may render some asm constraints impossible when compiling for PIC. There are several possible solutions to this class of problems: Non-PIC-safe assembler should be properly marked as such by using the builtin define __PIC__. Examples in the OpenBSD ports tree are:

• emulators/xnp2

• multimedia/avidemux

• security/aircrack-ng

Some software even ships PIC-safe versions of its assembler code that just had to be enabled on OpenBSD:

• emulators/dosbox

The compiler flag -fomit-frame-pointer can be used to free up an additional register (%ebp) and alleviate register pressure if necessary:

• emulators/mupen64plus/video-glide64

• emulators/openmsx

• graphics/rawstudio

• x11/mplayer

Finally, some ports can be easily adapted to use PIC-safe assembly (saving/restoring %ebx across a cpuid call):

• games/0ad

• games/megaglest

Static PIE

Recent work on PIE in OpenBSD has focused on enabling static binaries to take advantage of ASLR. Since those binaries should ideally stay self-hosting, the relocation needs to be performed in the startup code of the executable itself. From the already existing self-relocation code in ld.so[15], a new variant of the C runtime (crt0.o) was developed: A slightly modified version of ld.so's mostly machine-independent _dl_boot_bind()[16] is now called by the machine-dependent startup code.[17] Further adjustments to integrate this support into the toolchain were necessary: Given the flag combination '-static -pie', the linker now creates binaries with the DYNAMIC flag set, but no PT_INTERP section in order to identify them as static PIE. Likewise, gcc will link them with the new

15. See src/libexec/ld.so/boot.c.

16. src/lib/csu/boot.h.

17. src/lib/csu/*/md_init.h, MD_RCRT0_START().

self-relocating rcrt0.o upon seeing these flags. Static PIE is supported on every architecture that runs as PIE by default.

ASLR in other operating systems

Linux has included a weak form of ASLR since the 2.6.12 kernel release. At the time of writing, many mainstream distributions enable this support by default (Ubuntu[18], Fedora[19], Debian[20], Arch[21]) and some even implement a model similar to OpenBSD where -fpie is the compiler default (Hardened Gentoo[22], Alpine Linux[23]) or are on the way (OpenSUSE[24]). The distributions who do not follow the default-PIE approach typically have a list of high-risk applications that are protected by PIE.

In addition to the much reduced PIE coverage, there are also known weaknesses in Linux' ASLR implementation, as exploited by the recently-published offset2lib attack:[25] While the first object is loaded at a random location on each execution, the subsequent objects are then loaded in sequential order without any additional randomisation. This weakness means that an address leak of any object (e.g. the main program when using PIE) discloses the complete memory layout of the application code. Mitigations have been proposed by the authors themselves, but have not yet been merged into the mainline kernel.[26]

Large patchsets such as PaX and grsecurity are often advertised as the solution to the poor state of Linux ASLR. However, with its state as an external patchset that has been maintained for over a decade and lack of adoption in distributions with a reasonably large userbase, it remains a poorly-tested configuration – according to the official website, even tests on non-x86 hardware are lacking.[27]

FreeBSD does not yet have ASLR or PIE support; however, there is work in progress.[28] Also, build infrastructure support has already been committed[29] using an approach different from that of OpenBSD: Instead of a compiler-integrated default, FreeBSD uses build flags (in this case, via bsd.prog.mk). The problem with this is obvious even in the initial commit, since now any binary that is linked to a static library cannot be PIE. At the least, the default for static libraries would need to be switched as well.

NetBSD, from which FreeBSD's implementation in progress derives, has added support for PIE and ASLR in 2007.[30] However, it must be enabled on a per-program basis by the user.[31]

Mac OS X has added support for ASLR in version 10.5 and enabled PIE by default when targeting version 10.7.[32] Additionally, the kernel is relocated to a random position on every boot as well since version 10.8.[33]

Microsoft Windows has included support for ASLR since Windows Vista[34] and has since steadily improved the feature.[35] Since Visual Studio 2010, it is the default (opt-out instead of opt-in).[36]

Possible future improvements

ASLR is not perfect. Notably, a process cloned by fork(2) will have the same address space layout as the original – a fact that has already been successfully exploited using a technique called 'Blind ROP' in order to bruteforce stack canaries and then find gadget addresses by overwriting the return address.[37] Such attacks are best defended against by employing a fork+exec pattern instead of simple fork(2)s. For OpenBSD's imsg-style daemons (relayd, httpd, snmpd, iked), a prototype already exists. However, third-party applications are likely to remain vulnerable unless upstream authors decide to take ASLR into account.

18. https://wiki.ubuntu.com/Security/Features
19. https://fedoraproject.org/wiki/Security_Features_Matrix
20. https://wiki.debian.org/Security/Features
21. https://wiki.archlinux.org/index.php/DeveloperWiki:Security#PIE
22. http://wiki.gentoo.org/wiki/Hardened/Toolchain
23. https://www.alpinelinux.org/about/
24. https://bugzilla.suse.com/show_bug.cgi?id=912298
25. Marco-Gisbert & Ripoll (2014).
26. Last version of the patch as of Feb 13 2014: http://marc.info/?l=linux-kernel&m=142065168226565&w=2
27. http://pax.grsecurity.net/
28. https://bugs.freebsd.org/bugzilla/show_bug.cgi?id=181497
30. http://cvsweb.netbsd.org/bsdweb.cgi/src/sys/kern/kern_pax.c?rev=1.18&content-type=text/x-cvsweb-markup
31. http://netbsd.gw.com/cgi-bin/man-cgi?security+7+NetBSD-current
32. https://developer.apple.com/library/mac/qa/qa1788/_index.html
33. http://movies.apple.com/media/us/osx/2012/docs/OSX_MountainLion_Core_Technologies_Overview.pdf
34. https://msdn.microsoft.com/en-us/library/bb430720.aspx
35. https://media.blackhat.com/bh-us-12/Briefings/M_Miller/BH_US_12_Miller_Exploit_Mitigation_Slides.pdf
36. https://msdn.microsoft.com/en-us/library/bb384887%28v=vs.100%29.aspx
37. Bittau et al. (2014).

Conclusion

OpenBSD's implementation has shown that it is a feasible strategy to deploy PIE by default. Any arising issues both in the base system and the ports collection have been few and comparatively easy to solve. With the ever-growing sophistication of ROP attacks, other operating systems are urged to take inspiration from OpenBSD's experience and use a similar model to protect their users. As it stands today, the open-source community is mostly behind proprietary software vendors and exploit writers in the ROP arms race.

Literature

- Bittau, Andrea; Belay, Adam; Mashtizadeh, Ali; Mazières, David; Boneh, Dan, Hacking Blind, in: Proceedings of the 2014 IEEE Symposium on Security and Privacy, Washington (2014), pp. 227–42.
- Buchanan, Erik; Roemer, Ryan; Shacham, Hover; Savage, Stefan, When Good Instructions Go Bad: Generalizing Return-Oriented Programming to RISC, in: Proceedings of the 15th ACM conference on Computer and communications security, New York (2008), pp. 27–38.
- Checkoway, Stephen; Davi, Lucas; Dmitrienko, Alexandra; Sadeghi, Ahmad-Reza; Shacham, Hover; Winandy, Marcel, Return-Oriented Programming without Returns, in: Proceedings of the 17th ACM conference on Computer and communications security, New York (2010), pp. 559–72.
- Marco-Gisbert, Hector; Ripoll, Ismael, On the Effectiveness of Full-ASLR on 64-bit Linux, presented at DeepSEC (2014).
- Roemer, Ryan; Buchanan, Erik; Shacham, Hover; Savage, Stefan, Return-Oriented Programming: Systems, Languages, and Applications, in: ACM Transactions on Information and System Security (TISSEC) 15 (2012), pp. 1–36.
- Shacham, H., The geometry of innocent flesh on the bone: return-into-libc without function calls (on the x86), in: Proceedings of the 14th ACM conference on Computer and communications security, New York (2007), pp. 552–61.

29. https://svnweb.freebsd.org/base?view=revision&revision=267233