# Secure Programming with GCC and GLibc

Marcel Holtmann

CanSecWest 2008, Vancouver

# Introduction

- Working for the Open Source Technology Center at Intel
- Used to work for the Red Hat Security Response Team
- Have been to CanSecWest once or twice ;-)

# Agenda

- Secure programming in general
- Welcome to 21st century
- Tips and tricks

# Secure programming

- Understand the limits and flaws of your programming language
- Understand your own code
- Expect the unexpected
- Do code reviews
- Listen to your compiler

# Programming languages

- C and C++ are not secure languages
- Go for Java, C# or similar languages
- But ask yourself which language has been used to write JVM for example

- There is always a weakest link

# Something to keep in mind

- You have to know what you are doing
- Programming is art

- Nothing I gonna tell you in the next 30 minutes is going to change this
- However it might make your life easier

# The threats

- Format string attacks
- Buffer overflows
- Heap overflows and double free
- Stack overwrites
- ELF section overwrites
- Fixed address space layout

# The protection

- The Linux kernel (if you use Linux)
- GCC compile time options
- GLibc runtime options

- And of course the developer

# Linux kernel options

- Address space layout randomization (ASLR)
  - mmap, Stack, vDSO as of 2.6.18
  - Heap/executable as of 2.6.24
  - Requirement for -pie
- ExecShield
  - NX emulation (Red Hat and Fedora only)
- Stack Protector

# GCC options

- gcc -fstack-protector
- ld -z relro
- ld -pie / gcc -fPIE
- gcc -D_FORTIFY_SOURCE=2 -O2
- gcc -Wformat -Wformat-security

# GLibc options

- Heap protection
- Double free checking
- Pointer encryption

- Enabled by default

# Distributions

- Every major Linux distribution will try to enable most of these “security” features
- Some patch the default options of GCC
- Normally they never contribute back to the upstream project
- Have options for these features and make the distributions use them

# Format strings

- -Wformat
  - Check format types and conversations
  - Safe to use and part of -Wall
- -Wformat-security
  - Check potential security risks within printf and scanf
  - Non string literals or missing format arguments
- Listen to compiler warnings

# Buffer checks

- -D_FORTIFY_SOURCE=2 -O2
  - During compilation most buffer length are known
  - Include compile time checks and also runtime checks
  - The source must be compiled with -O2
  - Format strings in writable memory with %n are blocked
- No negative impact has been reported
- Usage in upstream projects is almost zero

# Heap protection

- GLibc includes heap protection
- Double free attempts will be detected

- Always enabled when using GLibc
- No negative impact known

# Stack protection

- Mainline GCC feature
- Also known as stack smashing protection or stack canaries
- Missing support for ia64 and alpha systems

- Helps to reduce stack overflows, but a 100% protection can not be expected

# Randomization

- Position Independent Executable (PIE)
- Requires ASLR support in the kernel
- GCC and linker option (-fPIE and -pie)
- Doesn't work on hppa and m68k systems

- Randomization is limited and only good for protecting against remote vulnerabilities

# Pointer encryption

- Protection of pointer in writable memory
- It is hard, but in theory the randomization can be overcome
- Store only mangled function pointer and XOR with a random number
- Encryption is considered faster than canaries and as secure

# ELF protection

- Linker option (-z relro)
- Mark various ELF memory sections read-only before handing over the program execution
- Also known as ELF hardening or protection against GOT overwrite attacks
- No problem reported so far

# Red Hat and Fedora security

| | Fedora Core | | | | | | Fedora | | Red Hat Enterprise Linux | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 3 | 4 | 5 |
| | 2003Nov | 2004May | 2004Nov | 2005Jun | 2006Mar | 2006Oct | 2007May | 2007Nov | 2003Oct | 2005Feb | 2007Mar |
| Firewall by default | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y |
| Signed updates required by default | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y |
| NX emulation using segment limits by default | Y | Y | Y | Y | Y | Y | Y | Y | Y[2] | Y | Y |
| Support for Position Independent Executables (PIE) | Y | Y | Y | Y | Y | Y | Y | Y | Y[2] | Y | Y |
| Address Randomization (ASLR) for Stack/mmap by default[3] | Y | Y | Y | Y | Y | Y | Y | Y | Y[2] | Y | Y |
| ASLR for vDSO (if vDSO enabled)[3] | no vDSO | Y | Y | Y | Y | Y | Y | Y | no vDSO | Y | Y |
| Restricted access to kernel memory by default | | Y | Y | Y | Y | Y | Y | Y | | Y | Y |
| NX for supported processors/kernels by default | | Y[1] | Y | Y | Y | Y | Y | Y | Y[2] | Y | Y |
| Support for SELinux | | Y | Y | Y | Y | Y | Y | Y | | Y | Y |
| SELinux enabled with targeted policy by default | | | Y | Y | Y | Y | Y | Y | | Y | Y |
| glibc heap/memory checks by default | | | Y | Y | Y | Y | Y | Y | | Y | Y |
| Support for FORTIFY_SOURCE, used on selected packages | | | Y | Y | Y | Y | Y | Y | | Y | Y |
| All packages compiled using FORTIFY_SOURCE | | | | Y | Y | Y | Y | Y | | | Y |
| Support for ELF Data Hardening | | | | Y | Y | Y | Y | Y | | Y | Y |
| All packages compiled with stack smashing protection | | | | | Y | Y | Y | Y | | | Y |
| SELinux Executable Memory Protection | | | | | | Y | Y | Y | | | Y |
| glibc pointer encryption by default | | | | | | Y | Y | Y | | | Y |
| FORTIFY_SOURCE extensions including C++ coverage | | | | | | | | Y | | | |

*[1] Since June 2004, [2] Since September 2004, [3] Selected Architectures*

# Debian and Ubuntu security

- Install the Hardening wrapper
  - apt-get install hardening-wrapper
- Set an environment variable to activate it
  - export DEB_BUILD_HARDENING=1
  - export DEB_BUILD_HARDENING_[feature]=0
- Ubuntu has stack protector by default

# A trivial example

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <stdint.h>
#include <string.h>
#include <stdint.h>
#include <inttypes.h>

int main(int argc, char *argv[])
{
    char buf[16];

    if (argc > 1) {
        strcpy(buf, argv[1]);
        printf("Your first argument was: ");
        printf(buf);
        printf("\n");
    } else {
        fprintf(stderr, "Usage: %s ARG\n", argv[0]);
        exit(1);
    }

    return 0;
}
```

# Using the wrapper

```
# DEB_BUILD_HARDENING=1 make trivial
cc     trivial.c   -o trivial
trivial.c: In function ‘main’:
trivial.c:16: warning: format not a string literal and no format arguments


# ./trivial $(perl -e 'print "A"x100')
Your first argument was:
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAA
*** stack smashing detected ***: ./trivial terminated
Segmentation fault (core dumped)
```

# Other useful tools

- Statical analysis
- The Linux kernel sparse
  - User/kernel pointer checks
  - Endian conversion checks
- The memory checker valgrind
  - Listen to its warnings

# Conclusion

- Use the security features that are available and make them mandatory
- Listen to your compiler and understand the warnings – fix the cause, not the warning

- You still have to write good and secure code, but listen to your tools when they try to tell you something ...

# Thanks for your attention

marcel@holtmann.org