# Protection measures in OpenBSD

Matthieu Herrb & other OpenBSD developers

LAAS-CNRS

Université
de Toulouse

Coimbra, November 2009

# Agenda

1 Introduction

2 System level protection

3 Network level protection

4 What’s missing

5 Conclusion

# Agenda

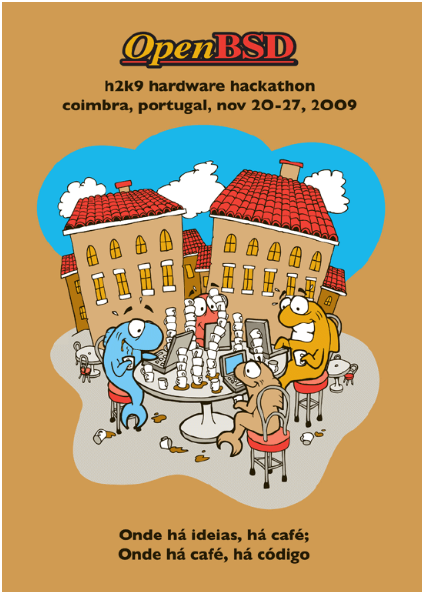
1 Introduction

2 System level protection

3 Network level protection

4 What’s missing

5 Conclusion



OpenBSD

h2k9 hardware hackathon
coimbra, portugal, nov 20-27, 2009

Onde há ideias, há café;
Onde há café, há código

# OpenBSD...

- Unix-like, multi-platform operating system
- Derived from BSD 4.4
- Kernel + userland + documentation maintained together
- 3rd party applications available via the ports system
- One release every 6 months
- Hardware architectures: i386, amd64, alpha, arm, macppc, sparc, sparc64, sgi, vax...

# Objectives

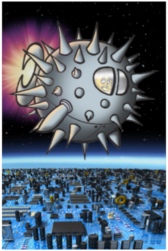
- Provide free code (BSD license...)
- Quality
- Correctness
- Adhering to standards (POSIX, ANSI)
- Providing good crypto tools (SSH, SSL, IPSEC,...)

→ better security.

# Current version

OpenBSD 4.6 released Oct 18, 2009.
New stuff :

- smtpd, a new privilege separated SMTP daemon
- tmux, a terminal multiplexer
- more sparc64 frame-buffers
- virtual routing and firewalling through routing domains
- routing daemon enhancements
- active-active firewall setups with pfsync
- ...



# Agenda

1. Introduction
2. System level protection
3. Network level protection
4. What’s missing
5. Conclusion

# “Secure by default”

- Leitmotiv since 1996
- Adopted since by most OS
- Non required services are not activated in a default installation.
- Default configuration of services providing security
- Activating services require a manual action of the administrator
- Keep a working (functional, useful) system

→ only 2 remote vulnerabilities in more than 10 years.

# Coding rules

- Focus on code correctness → improves reliability, thus security.
- Always design things for simplicity
- Peer review at every level : design, coding, even for simple modifications
- Comprehensive search for similar errors once one is found
- New functionalities in gcc :
  - `-Wbounded` option
  - `__sentinel__` attribute
- Tools : llvm/clang, Parfait, etc.

# Technologies for security

- `strlcpy/strlcat`
- memory protection
- privilege revoking (ex. `ping`)
- privileges separation (ex. OpenSSH)
- `(chroot)`
- separate uids for each service
- stack smashing protection (SSP) & Stackgap
- use of randomness (ld.so, malloc, mmap)

# Stack and memory protection

Stack overflows : the easiest exploitable vulnerability

- Stackgap
- GCC + Propolice (SSP) activated by default to build all libraries and applications.
- Generalized use of protection against format errors in printf()-like functions..

# Propolice

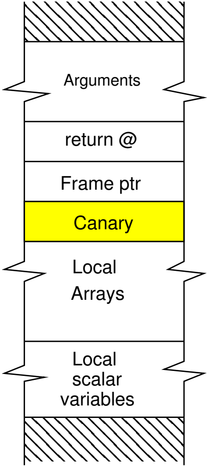
`http://www.trl.ibm.com/projects/security/ssp/`

Principle : put a “canary” on the stack, in front of local variables

- check it before return.
- if still alive: no overflow
- if dead (overwritten): overflow → `abort()`

Only when there are arrays in local variables

Adopted by gcc 4.1.



# W^X

Write **exclusive or** execution granted on a page..

- easy on some architectures (x86_64, sparc, alpha): per page ’X’ bit
- harder or others (x86, powerpc):
  per memory segment ’X’ bit
- impossible in some cases (vax, m68k, mips)

(PAX on Linux...)

# Random numbers in OpenBSD

A good random numbers source is important for security.

Entropy gathering :

- from I/O: keyboard, mouse, network or audio cards, etc.
- from hardware sources (VIA CPUs, crypto chips)

Use:

- Pseudo-random numbers using `arc4random()` to not drain entropy to fast.
- Centralized system → raises security. (harder to observe or predict).

# Randomness in the run-time linker

OpenBSD's `ld.so` loads libraries randomly in memory

Thus:
Random loading address for every shared object from one system to another
→ return to libc attacks are a lot more difficult.

# Randomness in mmap()

Address returned by `mmap()`:

If `MAP_FIXED` is not specified: returns a random address.

(traditional behaviour: 1st free page after a base starting address)

# Randomness in malloc()

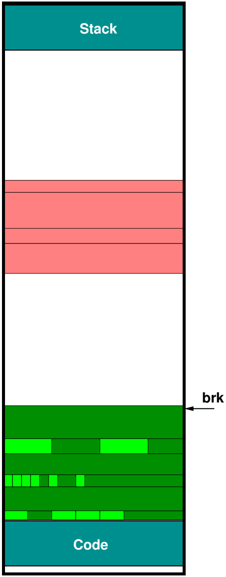
- $\geqslant 1$ page allocations: mmap() $\longrightarrow$ random addresses.
- $< 1$ page allocations: classical fixed block allocator, but random selection of the block in the free list.

$\Rightarrow$ heap attacks more difficult.

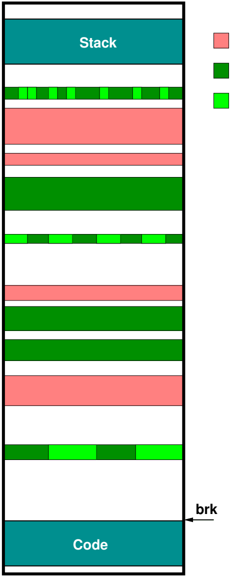
# Protecting dynamically allocated memory

[Moerbeek 2009]

- Activated by `/etc/malloc.conf` → `G`
- Each bigger than one page allocation is followed by a guard page ⇒ segmentation fault if overflow.
- Smaller allocations are randomly placed inside one page.



Traditional model



OpenBSD model

🟥 mmap()

🟩 malloc() >= 1page

🟩 malloc() < 1page

# Privileges reduction

- Completely revoke privileges from privileged (setuid) commands, or commands launched with privileges, once every operation requiring a privilege are done.
- Group those operations as early as possible after start-up. Examples:
  - ping
  - named

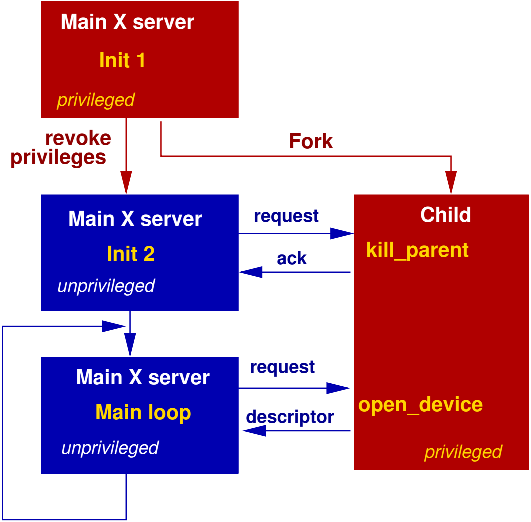
# Privileges separation

[Provos 2003]

- Run system daemons:
  - with an uid $\neq 0$
  - in a chroot(2) jail
- additional helper process keeps the privileges but do paranoid checks on all his actions.

A dozen of daemons are protected this way.

# Example : X server



Main X server

Init 1

*privileged*

revoke privileges

Fork

Main X server

Init 2

*unprivileged*

request

ack

Child

kill_parent

Main X server

Main loop

*unprivileged*

request

descriptor

open_device

*privileged*

# Securelevels

No fine grained policy:
too complex, thus potentially dangerous.

Three levels of privileges

- kernel
- root
- user

Default securelvel = 1:

- File system flags (immutable, append-only) to limit root access.
- Some settings cannot be changed (even by root).
- Restrict access to `/dev/mem` and raw devices.
- Exception : X...

# Agenda

1. Introduction
2. System level protection
3. Network level protection
4. What’s missing
5. Conclusion

# Threats on protocols

Internet: favours working stuff over security.

- easy to guess values
- forged packets accepted as valid
- information leaks
- use of time as a secret ??

# Protection Principle

Use data that are impossible (hard) to guess wherever arbitrary data are allowed, even if no known attack exists.

- counters
- timestamps
- packet, session, host... identifiers

Respecting constraints and avoid breaking things:

- non repetition
- minimal interval between 2 values
- avoid magic numbers

# Example : TCP Reset

“Slipping in the window”, Paul T. Watson (2004)

Resizing of TCP windows makes it easier to figure out a valid TCP sequence number.

Target : long living TCP connections, between machines that only have a few different connections open (BGP sessions for instance).

OpenBSD solution:

- really random source ports
- require that RST packets are at the extreme right of the window
- and of course also : TCP MD5 and/or IPsec protection. (OpenBGPD rejects TCP window negotiation without one of those).

# DNS

Security based on:

- (ip source, port source, ip dest, port dest)
- 16 bits identifier

OpenBSD:

- pseudo-random identifiers since 1997
- random source port

# Randomness in the network stack

Use:

- IPID (16 bits, no repetition)
- DNS Queries (16 bits, no repetition)
- TCP ISN (32 bits, no repetition, steps of $2^{15}$ between 2 values)
- Source ports (don’t re-use a still active port)
- TCP timestamps (random initial value, then increasing at constant rate)
- Id NTPd (64 bits, random) instead of current time
- RIPd MD5 auth...

# PF: more than one trick in its bag

Packet Filter

- Stateful filtering and rewriting (NAT) engine
- **Scrub** to add randomness to packets:
  - TCP ISN
  - IP ID
  - TCP timestamp
  - NAT : rewriting of source ports (and possibly addresses)

Also protects non-OpenBSD machines.

# Agenda

1. Introduction
2. System level protection
3. Network level protection
4. What’s missing
5. Conclusion

# User application protection

- Web browsers,
- Multimedia viewers and players,
- E-mail clients,

Not protected enough. Hint : privilege separation / sandboxing (Chrome OS)

Problem : how to handle sophisticated social engineering attacks ?

# X Windows problems

Kernel-level privileged code running in user-space. vulnerabilities in X are thus especially critical. (cf. Loïc Duflot Cansecwest)

- Privilege separation (not enough, unfortunately)
- Kill all direct hardware register access : vesafb
- Future : KMS + DRI : programming the hardware in the kernel, userland access controlled by DRI.

# Agenda

1 Introduction

2 System level protection

3 Network level protection

4 What’s missing

5 Conclusion

# Conclusion

- Lots of progress since the beginning.
- Contributed to fix bugs in many 3rd party applications.
- Copied often (good).
- Still lots of issues to address...
- Will it be finished someday ?

# Bibliography

`http://www.openbsd.org/papers/index.html`

- *A new malloc(3) for OpenBSD*, Otto Moerbeek EuroBSDCon 2009, Cambridge.
- *Using OpenBSD Security Features to Find Software Bugs*, Peter Valchev, Reflections/Projections, Champaign-Urbana, 2007
- *Time is not a secret: Network Randomness in OpenBSD*, Ryan McBride Asia BSD Conference 2007
- *Security Measures in OpenSSH*, Damien Miller Asia BSD Conference 2007
- *The OpenBSD Culture*, David Gwyne : OpenCON 2006
- *Security issues related to Pentium System Management Mode*, Loïc Duflot, CansecWest 2006.
- *Exploit Mitigation Techniques*, Theo de Raadt OpenCON 2005, Venice, Italy
- *A Secure BGP Implementation*, Henning Brauer SUCON 04
- *Preventing Privilege Escalation*, Niels Provos, Markus Friedl and Peter Honeyman, 12th USENIX Security Symposium, Washington, DC, August 2003.
- *Enhancing XFree86 security*, Matthieu Herrb LSM, Metz 2003.

Questions ?