Unusual bugs

Ilja van Sprundel
ilja@suresec.org

Who am I

- Loves to see things crash
- Suresec
- Netric
- http://blogs.23.nu/ilja
- http://ilja.netric.org/

agenda

- NULL pointers
- stack overflow
- Shifting the stackpointer
- Reference counters
- File race conditions
- Regular expressions
- Writing to stderr

Introduction

- No big secrets
- Mentioned somewhere already
- Not that many people seem to know about them
- I bumped into most of them when doing code reviews
- Most of it is ongoing research

NULL pointers

“It seems that the combination of a very large input and a crash leads some people to automatically think the problem may be used to "execute arbitrary code", that is, take over the user's computer. That is not always the case, and is usually only possible for what is called a stack buffer overflow, even though it is also sometimes possible for heap (allocated) buffer overflows. However, we have seen this phrase used even for **ordinary NULL pointer crashes that had no possibility whatsoever to be abused.**”

Yngve Pettersen
Senior developer Opera Software

NULL pointers

- Most are considered not to be exploitable
- Because 0x00000000 is not mapped
- Sometimes you don’t need it to be mapped
- NULL + <some offset> for example
  - Where offset is something large you control
- Pass NULL to a function where it has special meaning
  - Strtok() for example

NULL pointers cont.

- NULL + offset example (procmail):

```
void*app_val_(sp,size)
struct dyna_array*const sp;
int size;
{
  if(sp->filled==sp->tspace) /* need to grow ? */
   {
     size_t len=(sp->tspace+=4)*size;
     sp->vals=sp->vals?realloc(sp->vals,len):malloc(len);
   }
  return &sp->vals[size*sp->filled++];
}
```

NULL pointers cont.

```
#define app_val_type(sp,t,s,v)    (*(t*)app_val_(&sp,sizeof(s))=(v))
#define app_valp(sp,val)          app_val_type(sp,const char*,const char*,val)
```

- Yes, procmail is nasty
- If realloc() fails at some point it returns:
  - NULL + sp->filled++
  - NULL + offset
  - And then writes a userdefined byte to it

NULL pointers cont.

- NULL has special meaning example (X.org):

```
char            *tok_lasts;
char            *spnamelist;
char            *spname;
spnamelist =
                strdup(XpGetSpoolerTypeNameList());
/* strtok_r() modifies string so dup' it first */
for( spname = strtok_r(spnamelist, ":", &tok_lasts) ;
    spname != NULL ;
    spname = strtok_r(NULL, ":", &tok_lasts) )
{
```

NULL pointers cont.

- strdup() can return NULL
  - If there‘s not enough memory
- So NULL is passed to strtok_r
- NULL as 1st arg to strtok_r has special meaning
- It mean: “I’ve called strtok_r() before, use the 3th arg”
- Which in this case is an uninitialized pointer to a pointer
- Strtork_r then scans through the string till it finds the 2nd arg
- And then replaces it with a 0-byte ! (causing memory corruption)

NULL pointers cont.

- NULL is just another address
- Fits perfectly into your virtual address space
- But unmapped by default in most cases
- If only you could get it mapped …
  - If so, a lot of NULL pointer bugs become exploitable
- So in which context could you map NULL
  - Before it dereferences NULL in some way

NULL pointers cont.

- Mapped by default.
- Some osses used to map it by default.
- You could adleast read 0x00000000
- There was some data on it
- 8lgm made an exploit for a NULL pointer deref SCO’s pt_chmod

NULL pointers cont.

```
/*
 * ptchown.c
 * Usage: ptchown file
 * Copyright [8lgm] 1994, all rights reserved.
 *
 * Utilises flaw in /usr/lib/pt_chmod to chown 'file' to your uid/gid.
 * If invoked with a fd that isnt a valid master pty, ptsname() will
 * return NULL, which goes unchecked.  pt_chmod will thus execute:
 *
 *         chown(NULL, getuid(), getgid());
 *
 * If your OS maps page 0, such that NULL is a valid address,
 * the chown(2) will succeed.  This program works by symlinking
 * 'file' to NULL, and then executing pt_chmod with an 'invalid' fd.
 *
 * […]
 */
```

NULL pointers cont.

- The kernel !
- If in kernel mode, but user context
  - Virtual address is shared (for most osses anyway)
- So how does one map NULL ?
- mmap(0,<length>, PROT_READ|PROT_WRITE, MAP_FIXED|MAP_PRIVATE, -1, 0);
- Ofcourse exploitability still depends
  - What the kernel does with this NULL pointer
- Still, a fair amount of kernel NULL pointers can be exploited this way

NULL pointers cont.

- So what about userland ?
- Fairly os dependant.
- The goal is still the same
  - For a process to map 0x00000000 somehow
- Most of my testing was done on linux

NULL pointers cont.

- In early 2.6.x kernels (till 2.6.12)
- Mmap allocates from high till low addresses
  - 0xbfXXXXXX till 0x00000000
  - Including 0x00000000 !!!
- Get an application to map a lot of memory
- Eventually 0x00000000 gets mapped !
- Think firefox ☺

NULL pointers cont.

commit 49a43876b935c811cfd29d8fe998a6912a1cc5c4
Author: Linus Torvalds <torvalds@osdl.org>
Date: Wed May 18 15:39:33 2005 -0700

[PATCH] prevent NULL mmap in topdown model

Prevent the topdown allocator from allocating mmap areas all the way down to address zero.

We still allow a MAP_FIXED mapping of page 0 (needed for various things, ranging from Wine and DOSEMU to people who want to allow speculative loads off a NULL pointer).

Tested by Chris Wright.

Signed-off-by: Linus Torvalds <torvalds@osdl.org>

NULL pointers cont.

- Another way to force local applications to map NULL
- Linux has this notion of personalities
- One of them (SVr4) requires NULL to be mapped by default
- Personalities get inherited thru execve !

NULL pointers cont.

- binfmt_elf.c:

```
if (current->personality & MMAP_PAGE_ZERO) {
        /* Why this, you ask???  Well SVr4 maps page 0 as read-only,
           and some applications "depend" upon this behavior.
           Since we do not have the power to recompile these, we
           emulate the SVr4 behavior.  Sigh.  */
        /* N.B. Shouldn't the size here be PAGE_SIZE?? */
        down_write(&current->mm->mmap_sem);
        error = do_mmap(NULL, 0, 4096, PROT_READ | PROT_EXEC,
                        MAP_FIXED | MAP_PRIVATE, 0);
        up_write(&current->mm->mmap_sem);
}
```

NULL pointers cont.

- A downside to this ‘feature’
- NULL gets mapped
  - Read only
  - Filled with 0-bytes
- Limits the options
- However, 0x0000 on ia32 can be seen as an alternative nop
  - Well, sort of, eax has to contain a valid address
- Still usefull for function pointers that are set to NULL

NULL pointers cont.

- Some osses also map NULL by default
- But map it without read, write or exec perms
- Mostly useless
  - Unless you can trigger a NULL pointer deref right before a call to mprotect()

Stack overflow

“I agree that recursion overflow doesn't look like a very promising area for vulnerability research.”

Michael Wojcik
Principal Software Systems Developer, Micro Focus

Stack overflow

- They’re considered to be nothing more then a DoS
- But if you think about it, it’s just another overflow.
- What if you can force an application to map data directly after the stack ?
- If it’s local, you can change rlimits:
  - Put arbitrary limits on the length of the stack
  - Force the memory allocator to use mmap instead of brk
- Again a question of context
- Highly os dependant

Stack overflow cont.

- Tested on linux 2.6.8 (debian)
- Mmap’d data grows towards the stack
- There is no guard page between stack and mmap’d data per se.
- If you can allocate enough memory stack and heap are right next to each other !

Stack overflow cont.

- So what about (p)threads ?
- Libc dependant
- With older glibc's there was no guard page between stacks
- Newer one's do have a guard page
- Sometimes you can jump over a guardpage !
- Consider:

```
void f(void) {
  char b[4096];
  f(); }
```

Stack overflow cont.

- What about the kernel ?
- Most don’t specifically have a guard page for all stacks.
- Huge waste of memory
- If you can force another page to be mapped after (or before, depends !) the stack then it becomes exploitable !

Shifting the stackpointer

Shifting the stackpointer

- There are certain ways to allocate dynamically sized arrays on the stack
- C99 provides the following:
```
f(int s){
  char b[s];
}
```
- And then there is alloca
- Quote from the alloca manpage:
  “The **alloca**() function is machine and compiler dependent. On many systems its implementation is buggy. Its use is discouraged.”

Shifting the stackpointer cont.

- Dynamically allocated arrays on the stack:
  - Fn(int len, char *p) {
    char b[len];
    strcpy(b,p);
    }
- Alloca:
  - Fn(int len, char *p) {
    char *b = alloca(len);
    strcpy(b,p);
    }

Shifting the stackpointer cont.

- Gcc generates code that simply modifies the stack pointer
- Generated code for alloca and C99 dynamically allocated arrays on the stack is pretty much identical
- No error checking of any kind is done

Shifting the stackpointer cont.

- Unbounded calls: pwned ! You just got full control over the stack pointer.
- Even if not unbounded. If the length is based on some stringlength for example it can still be fairly large.
- And cause stack overflow
- Depending on the context these are still exploitable (as mentioned earlier) !

Shifting the stackpointer cont.

- Most compilers I’ve seen don’t add any protection here
- Microsoft’s visual studio does !
  - Probes memory in alloca
  - Doesn’t implement C99 dynamically sized arrays

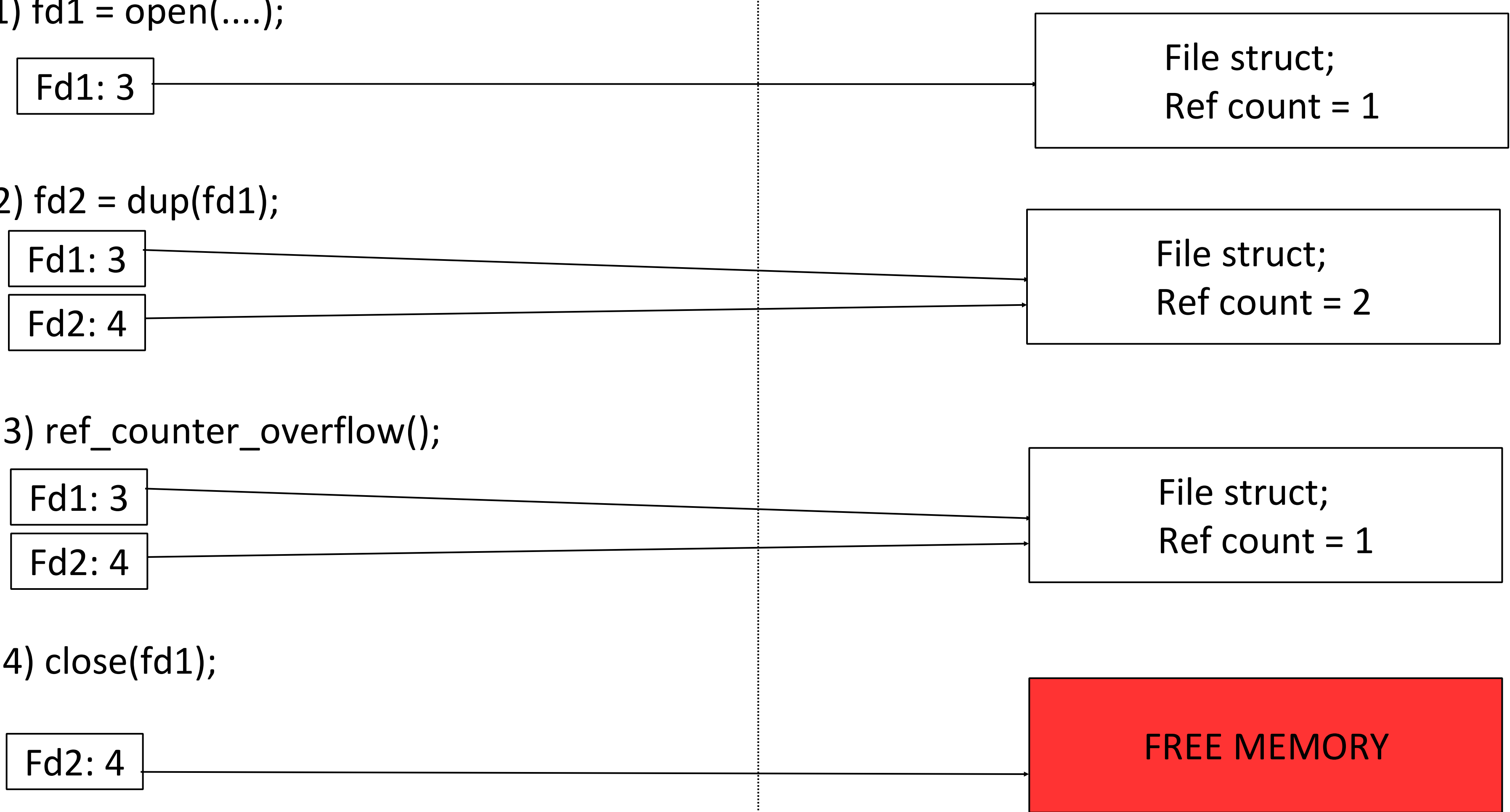
Reference counters

Reference counters

- Certain objects have reference counters
- Used to prevent free’ing that object while it’s in use
- If you use the object you increase the refcount
- When done you decrease it
- If the refcount is zero it get’s free’d

Reference counters cont.

- Turns out reference counting is more difficult then it seems
- Lots of cases where you might forget to decrease a reference (error code paths)
- Leads to an integer overflow of the reference counter
- Refcount goes to 3,4,5,….,0xffffffff,0,1



Userspace
Kernelspace
1) fd1 = open(....);
Fd1: 3
File struct;
Ref count = 1
2) fd2 = dup(fd1);
Fd1: 3
Fd2: 4
File struct;
Ref count = 2
3) ref_counter_overflow();
Fd1: 3
Fd2: 4
File struct;
Ref count = 1
4) close(fd1);
Fd2: 4
FREE MEMORY

Reference counters cont.

```
Int sys_fsync_range(struct lwp *l, void *v, register_t *retval) {
        struct sys_fsync_range_args  *uap = v;
        struct file *fp;
        int flags;
        int error;
        ...
        /* getvnode() will use the descriptor for us */
        if ((error = getvnode(p->p_fd, SCARG(uap, fd), &fp)) != 0)
        ...
        if (((flags & (FDATASYNC | FFILESYNC)) == 0) ||
           ((~flags & (FDATASYNC | FFILESYNC)) == 0)) {
                return (EINVAL); /*← no FILE_UNUSE(fp,l)*/
        }
```

Reference counters cont.

- Sometimes you also end up with a double decrease
- Easier to exploit
- Since you don’t have to wait for the integer overflow to happen
- Can be exploited instantly

Reference counters cont.

- Interesting suspects
  - Most kernels reference a lot of object (vm areas, files, …)
  - X server
  - Mozilla XPCOM
  - (D)COM(+)
  - Javascript (objects reference counted behind the scences)
  - Python
  - A whole lot more …

File race conditions

File race conditions

- Reopening in a file in /tmp in a (un)safe manner.
- Suggested solutions: “lstat/open/fstat dance”
  - Compare both devices and inodes
  - If they match it must be the same file
- Not true
- Still a race
  - File could be deleted
  - Create new file with the same inode nr

File race conditions cont.

- “But if you compare more from the stat struct it’s safe”
- Not true

File race conditions cont.

```
struct stat {
       dev_t st_dev; /* ID of device containing file */
       ino_t st_ino; /* inode number */
       mode_t st_mode; /* protection */
       nlink_t st_nlink; /* number of hard links */
       uid_t st_uid; /* user ID of owner */
       gid_t st_gid; /* group ID of owner */
       dev_t st_rdev; /* device ID (if special file) */
       off_t st_size; /* total size, in bytes */
       blksize_t st_blksize; /* blocksize for filesystem I/O */
       blkcnt_t st_blocks; /* number of blocks allocated */
       time_t st_atime; /* time of last access */
       time_t st_mtime; /* time of last modification */
       time_t st_ctime; /* time of last status change */
};
```

File raceconditions cont.

- `dev_t st_dev;`
  - So it has to be on the same devide
- `Ino_t st_ino;`
  - And have the same inode number
- `mode_t st_mode;`
  - And the same mode
- `nlink_t st_nlink;`
  - And the same amount of hard links

File Race conditions cont.

- `uid_t st_uid;`
  `gid_t st_gid;`
  - And owned by the same user and group
- `off_t st_size;`
  - And it has to have the same size
- `blksize_t st_blksize;`
  - Is usually the same for all files
- `blkcnt_t st_blocks;`
  - Amount of diskspace the file occupies (blocks of 512b)

File race conditions cont.

- `time_t st_atime;`
  `time_t st_mtime;`
  `time_t st_ctime;`
  - Time since epoch mesured in **SECONDS** !!!
  - You get a whole second to race.
  - Depending on how the fs is mounted atime doesn’t matter.
  - time_t can overflow, if you’re willing to wait for 70 years ☺

File race conditions cont.

- Normally people only compare a few fields, and you don’t have to make sure all of this is valid.

File race conditions cont.

- “O_NOFOLLOW prevents this race”
- No, it doesn’t
- It just prevents an attacker from using symlinks in the attack, there is still a race
- O_NOFOLLOW will still follow directory symlinks

File race conditions cont.

- Piled and piles of vulnerable applications !
- Go reaudit all those cool suids ☺

Regular expressions

Regular expressions

- Often used for input validation
- Or patternmatching for some kind of detection (snort for example)
- Regular expressions are exceptionally hard to get right
- Even those that look very simple

Regular expressions cont.

- For example:
  - /^\d\d\d-\d\d\d$/
  - “$” will also match \n
  - Could lead to next line injection

Regular expressions cont.

- How about this one:

```
(?:[\040\t]|\((?:[^\\\x80-\xff\n\015()]|\\[^\x80-\xff]|\((?:[^\\\x80-\xff\n\015(
)]|\\[^\x80-\xff])*\))*\))*(?:(?:[^(\040)<>@,;:".\\\[\]\000-\037\x80-\xff]+(?![^
(\040)<>@,;:".\\\[\]\000-\037\x80-\xff])|"(?:[^\\\x80-\xff\n\015"]|\\[^\x80-\xff
])*")(?:(?:[\040\t]|\((?:[^\\\x80-\xff\n\015()]|\\[^\x80-\xff]|\((?:[^\\\x80-\xf
f\n\015()]|\\[^\x80-\xff])*\))*\))*\.(?:[\040\t]|\((?:[^\\\x80-\xff\n\015()]|\\[
^\x80-\xff]|\((?:[^\\\x80-\xff\n\015()]|\\[^\x80-\xff])*\))*\))*(?:[^(\040)<>@,;
:".\\\[\]\000-\037\x80-\xff]+(?![^(\040)<>@,;:".\\\[\]\000-\037\x80-\xff])|"(?:
```

Regular expressions cont.

People make big regex’s:

“limit of GNU regex

I wrote a huge regex on GNU regex, it couldn't compiled.

I can find the limitation from the code:

* Max 255 pairs of ()

* Max 65535 bytes of regex string “

- This is insane !

Regular expressions cont.

- Some regex implementations are not binary safe either.
- The poison null byte strikes again
- Php’s ereg() functions have this problem
- Can be used in a lot of cases for all sort of nastyness (sql injection for example)

Regular expressions cont.

- Taking a regex as input
- Stupid imo, but people do it
- Javascript has support for regular expression
- Most regex parsers are written in c
- And hence most regex parsers are probably vulnerable to a lot of overflows and the like
- Perl regex's allow embedded perl code in a regex
  - Not allowed by default if the regex is not constant
  - "use re 'eval'" enables it

Writing to stderr

Writing to stderr

- Hm, well file descriptor 2
- In libraries
- Unexpectatly
- Example: FreeBSD malloc:
  - Writes to stderr if it can’t allocate some management structures
- What if fd 2 got closed ?
- And then for example /etc/passwd gets opened read-write

Writing to stderr cont.

```
static arena_t *
arenas_extend(unsigned ind)
{
        ...
        /*
         * OOM here is quite inconvenient to propagate, since dealing with it
         * would require a check for failure in the fast path.  Instead, punt
         * by using arenas[0].  In practice, this is an extremely unlikely
         * failure.
         */
        malloc_printf("%s: (malloc) Error initializing arena\n",
            _getprogname());
        ...
}
```

- malloc_printf() prints to stderr
- _getprogname() usually returns the content of argv[0]

Writing to stderr cont.

- Bugs where you can passed closed fd’s to suids obviously help
  - OSX
  - Opensolaris
  - Linux kernel (doesn’t work with glibc !)

Writing to stderr cont.

- Most library functions should never write to a fixed file descriptor (stderr for example)
- If they need to do so, then it should be documented …
- … or painfully obvious (printf() for example)

credits

- I didn’t do all of this on my own
- Talked to some people about various things
- Ronald Huizer
- Joel Eriksson
- Adam Shostack
- FX
- Many more I probably forgat …

Useful resources

- The shellcoders handbook (check page 505)
- http://eeyeresearch.typepad.com/blog/2006/08/post_ms06035_ma.html
- http://www.phrack.org/phrack/63/p63-0x0e_Shifting_the_Stack_Pointer.txt
- http://seclists.org/vuln-dev/2002/Nov/att-0056/0
- http://www.pine.nl/press/pine-cert-20030101.txt
- http://seclists.org/bugtraq/2000/Jan/0016.html

Useful resources cont.

- http://blogs.23.nu/RedTeam/stories/12189/
- http://www.snort.org/archive-3-1353.html
- http://gcc.gnu.org/ml/gcc-patches/2004-02/msg02289.html