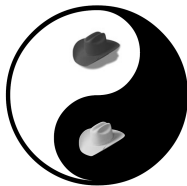


# Introduction to secure C programming

Michael Iatrou

Patras Linux User Group

December 17, 2008



```
/* Can't happen */
```

- Introduction
- Strings
- Integers
- Filesystem operations
- Random stuff
- Tools

# In the beginning...was UNIX

## The Good Books

- “The C Programming Language” by K&R
- “Advanced Programming in the UNIX Environment” by Stevens
- “Linux System Programming” by Robert Love

## Tools of the trade

- GNU/Linux
- gcc, glibc
- 32-bit Intel x86 (IA-32)

# Defining secure software

## Software qualities

- Trustworthiness
- Predictable execution
- Conformance

## Secure software properties

- Confidentiality
- Integrity
- Availability
- Accountability
- Non-repudiation

*"We will bankrupt ourselves in the vain search for absolute security."*

*– Dwight David Eisenhower*

# Security: Principles and flaws

## Principles

- Least privilege
- Fail-safe defaults
- Economy of mechanism
- Complete mediation
- Open design
- Separation of privilege
- Least common mechanism
- Psychological acceptability

## Flaws

- Input validation
- API abuse
- Security features
- Time and state
- Errors
- Code quality
- Encapsulation
- Environment

Validate **all** input.

*"In God we trust.  
All others must present a valid X.509 certificate."  
– Christos Ricudis*

# “Input” as in...

- Environment variables (IFS, LD\_LIBRARY\_PATH, LD\_PRELOAD, ...)
- Command line
- Graphical user interface (GUI)
- Files (contents, filenames, paths)
- File descriptors
- Network data

# Strings 101

- String operations in C are fast...and totally broken!
- Actually, C doesn't have "strings"
- String: a '\0' terminated byte stream
- `strlen()` returns the length of the string, **not** including the terminating '\0'
- `sizeof` can be used for statically allocated strings, returns the length of the string, including the terminating '\0'

# Unbounded string copy

```
int main(void) {
    char pass[64];
    puts("Enter password (64 characters max): ");
    gets(pass);
    ...
}
```

```
int main(void) {
    char passwd[65];
    puts("Enter password (64 characters max): ");
    fgets(passwd, 65, stdin);
    ...
}
```

# Unbounded string copy

```
int main(void) {
    char pass[64];
    puts("Enter password (64 characters max): ");
    gets(pass);
    ...
}
```

```
int main(void) {
    char passwd[65];
    puts("Enter password (64 characters max): ");
    fgets(passwd, 65, stdin);
    ...
}
```

# More unbounded string copy

## Broken

```
int main(int c, char *v[])
{
    char name[1024];
    int i = 0;
    while (v[i] != NULL)
        strcat(name, v[i++]);
    ...
}
```

## Fixed

```
int main(int c, char *v[])
{
    char *p;
    int sz = 0, i = 0;
    while (v[i] != NULL)
        sz += strlen(v[i++]);
    p = calloc(1, sz + 1);
    i = 0;
    while (v[i] != NULL)
        strcat(p, v[i++]);
    ...
}
```

# More unbounded string copy

## Broken

```
int main(int c, char *v[])
{
    char name[1024];
    int i = 0;
    while (v[i] != NULL)
        strcat(name, v[i++]);
    ...
}
```

## Fixed

```
int main(int c, char *v[])
{
    char *p;
    int sz = 0, i = 0;
    while (v[i] != NULL)
        sz += strlen(v[i++]);
    p = calloc(1, sz + 1);
    i = 0;
    while (v[i] != NULL)
        strcat(p, v[i++]);
    ...
}
```

# NULL termination

## Broken

```
char a[8], b[16];  
strncpy(a, "cafebabe", sizeof(a));  
strncpy(b, a, sizeof(a));
```

## Less broken

```
char a[8], b[16];  
strncpy(a, "cafebabe", sizeof(a));  
a[7] = '\\0';  
strncpy(b, a, sizeof(a));
```

# NULL termination

## Broken

```
char a[8], b[16];  
strncpy(a, "cafebabe", sizeof(a));  
strncpy(b, a, sizeof(a));
```

## Less broken

```
char a[8], b[16];  
strncpy(a, "cafebabe", sizeof(a));  
a[7] = '\\0';  
strncpy(b, a, sizeof(a));
```

# More NULL termination

## (almost) Fixed

```
char a[8], b[16];  
strcpy(a, "cafebabe", sizeof(a));  
strcpy(b, a, sizeof(b));
```

## Fixed

```
char a[8], b[16]; int i;  
strcpy(a, "cafebabe", sizeof(a));  
strcpy(b, a, sizeof(b));  
for (i = sizeof(a); i < sizeof(b); i++)  
    b[i] = '\0';
```

# More NULL termination

## (almost) Fixed

```
char a[8], b[16];  
strcpy(a, "cafebabe", sizeof(a));  
strcpy(b, a, sizeof(b));
```

## Fixed

```
char a[8], b[16]; int i;  
strcpy(a, "cafebabe", sizeof(a));  
strcpy(b, a, sizeof(b));  
for (i = sizeof(a); i < sizeof(b); i++)  
    b[i] = '\\0';
```

# Leave my string format alone!

```
void my_print(const char *str) {  
    printf(str);  
    return;  
}
```

```
void my_other_print(const char *str) {  
    printf("%s", str);  
    return;  
}
```

```
void my_kinda_fixed_print(const char *str, size_t n) {  
    printf("%.*s", n, str);  
    return;  
}
```

# Leave my string format alone!

```
void my_print(const char *str) {  
    printf(str);  
    return;  
}
```

```
void my_other_print(const char *str) {  
    printf("%s", str);  
    return;  
}
```

```
void my_kinda_fixed_print(const char *str, size_t n) {  
    printf("%.*s", n, str);  
    return;  
}
```

# Use “safe” string functions

<b>Unsafe</b>	<b>Less unsafe</b>
<code>strcpy()</code>	<code>strncpy()</code> , <code>strncpy()</code>
<code>strcat()</code>	<code>strlcat()</code> , <code>strncat()</code>
<code>strcmp()</code>	<code>strncmp()</code>
<code>strlen()</code>	<code>strnlen()</code>
<code>gets()</code>	<code>fgets()</code>
<code>strdup()</code>	<code>strndup()</code>
<code>sprintf()</code> , <code>snprintf()</code>	<code>asprintf()</code> , <code>vasprintf()</code>
<code>scanf()</code> , <code>sscanf()</code>	-
<code>getwd()</code>	<code>getcwd()</code>
<code>realpath()</code>	<code>canonicalize_file_name()</code>

# Integers 101

- Two's complement representation, sign is the MSB
- Two integer signedness types: signed, unsigned
- Standard signed integer types: signed char, short int, int, long int, long long int
- Extended types: int#\_t, uint#\_t, intptr\_t, uintptr\_t, intmax\_t, uintmax\_t
- Minimum and maximum values depend on: the type's representation, signedness, number of allocated bits
- The C99 standard sets minimum requirements for integer types
- The C99 defines implicit integer promotions/conversions

# Integer error conditions and checks

- Overflow
    - Signed or unsigned
    - Increased beyond its maximum value
    - Decreased beyond its minimum value
  - Sign error
    - Convert an unsigned integer to signed
    - Convert a signed integer to unsigned
  - Truncation
    - Convert to smaller integer value and original integer is out of the new range
- 
- Preconditions
  - Postconditions

# Integers: what could possibly go wrong?

```
int main(int c, char *v[])
{
    char *p;
    short int i = 0, sz = 0;

    while (v[i] != NULL)
        sz += strlen(v[i++]);
    p = calloc(1, sz + 1);
    i = 0;
    while (v[i] != NULL)
        strcat(p, v[i++]);
    ...
}
```

```
void *page_alloc(size_t pages)
{
    int psz = 4096;
    uint64_t sz;

    if (pages == 0)
        return NULL;
    sz = pages * psz;

    return (sz < UINT_MAX)
        ? calloc(psz, pages)
        : NULL;
}
```

# Integers: what could possibly go wrong, again?

```
char *
getstr(unsigned int l, char *s)
{
    unsigned int sz;
    char *d;

    sz = l - 2;
    d = malloc(sz + 1);
    memcpy(d, s, sz);

    return d;
}
```

```
int *table = NULL;

int table_ins(int pos, int v)
{
    if (!table)
        table = malloc(400);

    if (pos > 99)
        return -1;

    table[pos] = v;
    return 0;
}
```

# Temporary files

## Problem

File creation is error prone:

- May overwrite files
- May erase previous contents
- May expose data

## Solution

Create temporary file first, then rename it to final file:

- Must create temporary file on the same device as the final
- Temporary file needs unique filename

# Temporary file creation

```
int mkstemp(char *template)
```

- template: `"/path/to/file.XXXXXX"`
- permissions: `0600`
- returns: file descriptor

```
FILE *tmpfile(void)
```

- creates an anonymous file in `/tmp`
- returns: stream descriptor

**AVOID** `mktemp()`, `tempnam()`, `tmpnam()`

# Filesystem TOCTTOU flaws

## Broken

```
stat(filename, &st);
if (S_ISREG(st.st_mode)
    && st.st_size < 65536) {
    fd = open(filename, O_RDWR);
    ...
    close(fd);
}
```

## Fixed

```
fd = open(filename, O_RDWR);
if (fd != -1 &&
    fstat(fd, &st) != -1
    && st.st_size < 65536) {
    ...
}
close(fd)
```

# Filesystem TOCTTOU flaws

## Broken

```
stat(filename, &st);
if (S_ISREG(st.st_mode)
    && st.st_size < 65536) {
    fd = open(filename, O_RDWR);
    ...
    close(fd);
}
```

## Fixed

```
fd = open(filename, O_RDWR);
if (fd != -1 &&
    fstat(fd, &st) != -1
    && st.st_size < 65536) {
    ...
}
close(fd)
```

# TOCTTOU aware file renaming

```
int frename(int fd, const char *newname) {
    int n; char buf[64]; size_t nbuf2 = 256;
    sprintf(buf, "/proc/self/fd/%d", fd);
    char *buf2 = alloca(nbuf2);
    while ((n = readlink(buf, buf2, nbuf2)) == nbuf2)
        buf2 = alloca(nbuf2 *= 2);
    buf2[n] = '\\0';
    static const char deleted[] = " (deleted)";
    if (n < sizeof(deleted) || memcmp(buf2 + n
        - sizeof(deleted) - 1, deleted,
        sizeof(deleted) - 1) != 0)
        return rename(buf2, newname);
    errno = ENOENT;
    return -1;
}
```

# File name vs. file descriptor

<b>File name</b>	<b>File descriptor</b>
<code>stat()</code>	<code>fstat()</code>
<code>statfs()</code>	<code>fstatfs()</code>
<code>statvfs()</code>	<code>fstatvfs()</code>
<code>chown()</code>	<code>fchown()</code>
<code>chmod()</code>	<code>fchmod()</code>
<code>truncate()</code>	<code>ftruncate()</code>
<code>utimes()</code>	<code>futimes()</code>
<code>chdir()</code>	<code>fchdir()</code>
<code>execve()</code>	<code>fexecve()</code>
<code>setxattr()</code>	<code>fsetxattr()</code>
<code>getxattr()</code>	<code>fgetxattr()</code>
<code>listxattr()</code>	<code>flistxattr()</code>
<code>getfilecon()</code>	<code>fgetfilecon()</code>

# Changing directories

## Where am I?

```
1/  
1/2a/  
1/2a/3/  
1/2a/3/4/  
1/2b/ -> 2a/3/4
```

```
chdir("2b");  
chdir("../");
```

## Safe

```
int  
safe_chdir(const char *name) {  
    int dfd = open(name,  
                   O_RDONLY|O_DIRECTORY  
                   |O_NOFOLLOW);  
    if (dfd == -1)  
        return -1;  
    int ret = fchdir(dfd);  
    close(dfd);  
    return ret;  
}
```

# Changing directories

## Where am I?

```
1/  
1/2a/  
1/2a/3/  
1/2a/3/4/  
1/2b/ -> 2a/3/4
```

```
chdir("2b");  
chdir("../");
```

## Safe

```
int  
safe_chdir(const char *name) {  
    int dfd = open(name,  
                   O_RDONLY|O_DIRECTORY  
                   |O_NOFOLLOW);  
    if (dfd == -1)  
        return -1;  
    int ret = fchdir(dfd);  
    close(dfd);  
    return ret;  
}
```

# (Not so) random stuff

For non-cryptographic purposes:

- **AVOID** `rand()`, `rand_r()`
- Use `random()`, `random_r()`
- For compatibility use `rand48()`, `rand48_r()`
- Use a “good” seed: `gettimeofday()`, `getpid()`, `&seed`
- Poor man’s PRNG: `/dev/urandom`

Poor man’s crypto RNG: `/dev/random`

# Limit thyself!

```
int getrlimit(int resource, struct rlimit *rlim);
int setrlimit(int resource, const struct rlimit *rlim);

struct rlimit {
    rlim_t rlim_cur; /* Soft limit */
    rlim_t rlim_max; /* Hard limit (ceiling for rlim_cur) */
};
```

## “Interesting” resources to limit

- RLIMIT\_CORE
- RLIMIT\_DATA
- RLIMIT\_NOFILE
- RLIMIT\_NPROC
- RLIMIT\_SIGPENDING

# We could use some help...

## Static analysis

- `flawfinder`
- `RATS`
- `splint`

## Memory handling

- `valgrind`
- `mtrace`
- `dmalloc`
- `mudflap`
- `ElectricFence`
- `DUMA`

- Respect gcc warnings, use `-Wall -Wextra`
- Force failures, use `failmalloc`
- `gprof(1)` is your friend, `oprofile(1)` too
- Gather as much info from failures as possible (`catchsegv(1)`)

# Coverity scan: statistics for fun and for profit

Rung	Project	Fixed	Verified	Uninspected	LoC	Defects/KLoC
2	AMANDA	128	31	0	97,488	0.000
2	OpenVPN	0	1	13	69,667	0.014
2	Perl	46	1	91	497,724	0.030
2	Postfix	3	0	0	124,183	0.000
2	Python	77	2	6	284,926	0.004
1	apache-httpd	3	7	12	135,916	0.140
1	Firefox	370	56	246	1,860,622	0.162
1	gcc	32	15	171	851,149	0.219
1	glibc	83	0	0	588,931	0.000
1	KDE	1554	25	65	4,712,273	0.019
1	Linux-2.6	452	48	413	3,639,322	0.127
1	PostgreSQL	53	0	37	909,148	0.041
1	Subversion	25	2	8	154,468	0.065
1	X.org	502	3	0	25,583	0.117

<http://scan.coverity.com/> accessed on 2008-12-09

# Conclusions

- Security is not a feature, it's design and implementation discipline
- Be paranoid, don't trust anything you don't generate
- Assume stupidity, lack of knowledge and understanding (or just bad intent)
- Beware of dangerous implements, environment is volatile
- Never say "Can't happen"

# Thanks!

