3ème année

Sécurité des Systèmes Informatiques

SUPAERO

Rodolphe Ortalo
RSSI - CARSAT Midi-Pyrénées
rodolphe.ortalo@free.fr
(rodolphe.ortalo@carsat-mp.fr)
http://rodolphe.ortalo.free.fr/ssi.html

Overall presentation (1/2)

- Fast paced computer security walkthrough
  - Security properties
  - Attacks categories
  - Elements of cryptography
  - Introduction to mandatory security policies
- Embedded systems and security
  - Specificities
  - Physical attacks (SPA, DPA)
  - TPM
- **Software development and security**
  - **Security requirements and process**
  - Static verification and software development tools
  - Common criteria / ISO 15408

Introductory programmer comment

World-writable memory on Samsung Android phones

Posted Dec 17, 2012 20:13 UTC (Mon) by **mikov** (subscriber, #33179) [Link]

My experience from most places: nobody cares, nobody reviews. If a problem is discovered later, we will fix it later - why worry now and delay the release? What "/dev/mem"?? Enough with this mumbo-jumbo we have a release to make and management bonuses to earn.

In fact people who do care and worry about esoteric things like "security", or "good design" or "code quality" are universally viewed as trouble-makers or ivory tower idiots both by management and most of the engineers. It is an uphill battle even to do what used to be the baseline 10-15 years ago.

Commercial software engineering now is no different from accounting. The glory days are gone. It is all downhill from now on.

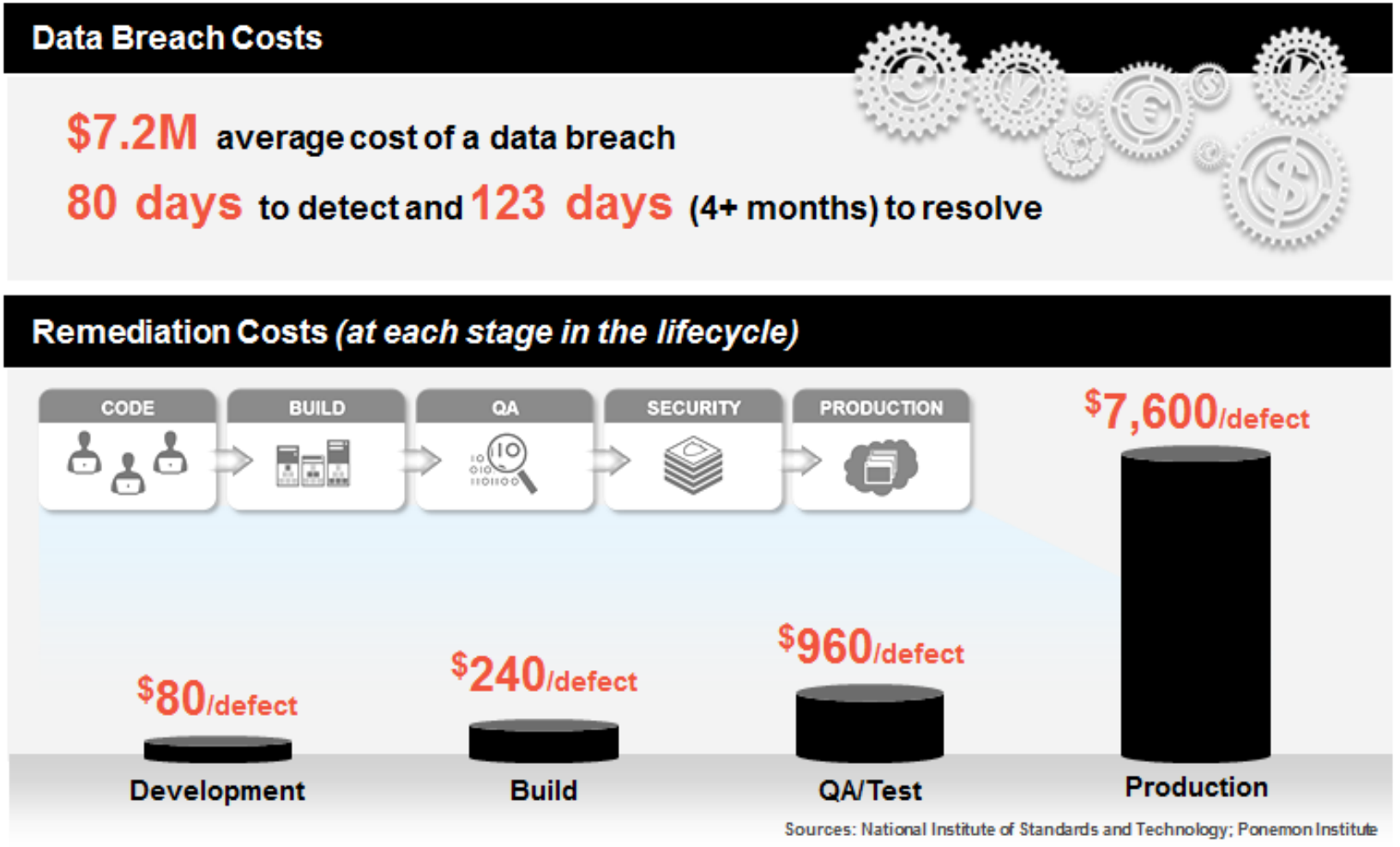
http://lwn.net/Articles/529496/

BTW, Cyanogen fix: http://review.cyanogenmod.org/#/c/28568/

Problem to address (with respect to security requirements definition)

- Best ROI when done at application design phase
- When considered at all, they tend to be
  - general lists of security features
    - password, firewalls, antivirus, etc.
  - implementation mechanisms ≠ security requirements
  - intented to satisfy *unstated* requirements
    - authenticated access, etc.
- Exist in a section by themselves (copied from a generic set)
  - no elicitation or analysis process, no adaptation to the target
- Significant attention is given to what the system should do
  - little is given to what it should not do *(in req. eng.)*
- Priority is not given to security (wrt ease of use for example)

Good old security stats. (w/pics)



Note on security updates

- How can we manage software vulnerabilities?
  - Wait until they are exploited by an attacker
  - Quickly provide a patch that should correct the problem (without introducing another one)
  - Whine because system administrators do not install patches fast enough
- Astonishingly it is very popular
  - All serious editors do that
  - Users feel more secure (still?)

Improving security Using Extensible Lightweight Static Analysis, David Evans and David Larochelle, *IEEE Software*, January/February 2002.

In other words

- It is not enough to apply patches to secure a system
- Also, you cannot rely only on firewalls or antivirus (or IT security tools)
- Security objectives of a piece of software should be identified
- Security implies a change in point of view
  - e.g.: it must *not* work
  - unavailable is better than destroyed
  - which (computer) is saved first ?

Another view on project lifecycle

- Detailed needs ? ⟷ Security policy consideration
- Specs ? ⟷ Security specification
- Contracts ⟷ Security specification
- Development
- Integration
- Validation ⟷ Security validation/configuration
- Exploitation → Monitoring / Management
- Maintenance → Monitoring / Management
- Disposal ? → Quid ?

Risk analysis

1. Identify assets and their value ($$)
2. Define assets priority
3. Identify vulnerabilities, threats and potential damages
4. Define threats priority
5. Optimize counter-measures selection

- Inherently qualitative (human/expert opinion)
- Applicable to organization, system, project
- Several methods available
  - MARION, MEHARI, EBIOS, etc.
  - HAZOP, FMEA, ISO31000, etc.

Pros (my view)

- *Identification* of assets and their relative values
- Assets value offers an opportunity to budget realistically (for protection)
- Is understandable by end users
  - Quite easier than assembly language exploits or cryptographic hash functions
- Risk management alternatives
  - Transfer (insurance, state, etc.)
  - Acceptance (life is deadly after all)
  - Reduction (work, work, work, work, ...)
  - Avoidance (just do it the other way)
- Management could express clear priorities

Cons (my view)

- Threat determination is an oracle problem
- May be used to demonstrate that (any) risk is (already) managed
  - Some forgotten successes of risk management
    - Lehman-Brothers financial risk exposure
    - Greek debt control
  - Qualitative also means manipulable
- Relies a lot on best practices or risks lists
  - Fuels paranoia and ready-made useless tools
  - Does not help target real assets
- Management rarely wants to decide
- Sometimes does not end well morally speaking
  - For example : product lifetime optimization
  - (NB : Inherently viewpoint-based)

Threats and use-case examples

- Trusted Computing Group
  - Mobile phone TPM use-case scenarios
  - *(Name,) Goal*
    - *Threats*

- Platform integrity
  - Ensure that device possess and run only authorized operating system(s) and hardware
    - Logic of device firmware modified
    - Device hardware modified
    - Device functions in a manner other than intended by the manufacturer
    - Device modified to broadcast false identification (IMEI)

Threats and goals examples

- Device authentication
  - Assist user authentication
  - Prove identity of device itself
    - Identity spoofing to get unauthorized access to services
    - Identity no longer bound to the device
    - Theft of device
    - Device tracking
- Robust DRM implementation
  - Service and content providers need assurance that the device DRM is robust
- SIMLock / Device personalisation
  - Ensure that a mobile device remains locked on a particular network

Last use-case examples (for info.)

- Secure software download
- Secure channel between device and UICC (UMTS Integrated Circuit Card)
- Mobile Ticketing
- Mobile Payment
- Software use
  - User-available predefined software use policies
- Proving platform and/or application integrity to end user
- User data protection and privacy

References

- DHS « Build Security In »
  - https://buildsecurityin.us-cert.gov/
- The Addison-Wesley Software Security Series
  - http://www.softwaresecurityengineering.com/series/
- CERT/CC
  - http://www.cert.org/
- « *Smashing the Stack for Fun and Profit.* »
  - Aleph One, Phrack Magazine 7, 49 (1996) File 14 of 16.
- OpenBSD
  - http://www.openbsd.org/papers/

Some real programming

- Presentation based on work from real programmers in the neighbourhood
- First, sources :
  - Matthieu Herrb & lots of OpenBSD « good programming » examples
  - Vincent Nicomette and Eric Alata for some « details »

Now real programming (*prereq*.)

```
#include <stdio.h>
void copie(char * s) {
  char ch[8] = "BBBBBBBB" ;
  strcpy(ch,s) ;
}
int main(int argc, char * argv[]) {
  copie(argv[1]) ;
  return(0);
}
```

AAAAAAAAAAAA[system_adr][exit_adr][shlibc_adr]

Bash$./a.out 'perl -e 'print "A"x12 . 0xb7ee1990 . 0xb7ed72e0 . 0xb7fcc0af" '

sh-3.1$

Now real programming

- Number One : buffer overflow with string functions

```
strcpy(path, getenv("$HOME"));
strcat(path, "/");
strcat(path, ".foorc");
len = strlen(path);
```

- strcat(), strcpy()
  - no verification on buffer size, dangerous : do not use
- strncat(), strncpy()
  - leave strings non terminated, very difficult to use correctly
- strlcat(), strlcpy()
  - May truncate strings, but probably easier to use

http://homepages.laas.fr/matthieu/cours/mh-prog-defensive.pdf

str{,n,l}{cpy,cat} practical usage

```
STRCAT(3)                  Linux Programmer's Manual                  STRCAT(3)

NAME
       strcat, strncat - concatenate two strings

SYNOPSIS
       #include <string.h>

       char *strcat(char *dest, const char *src);

       char *strncat(char *dest, const char *src, size_t n);
```

No strlcat() on Linux ; so, from the BSDs (more precisely OpenBSD) :

```
size_t strlcpy(char *dst, const char *src, size_t dstsize);
size_t strlcat(char *dst, const char *src, size_t dstsize);
```

strncat() is difficult to use

```
strncpy(path, homedir, sizeof(path) – 1) ;
path[sizeof(path) – 1] = '\0' ;
strncat(path, "/", sizeof(path) – strlen(path) – 1) ;
strncat(path, ".foorc", sizeof(path) – strlen(path)
    – 1) ;
len = strlen(path) ;
```

Note (on Linux) : g_strlcpy() and g_strlcat() exist in glib-2.0

Note (on BSD) : see next slide (*Yeah !!!)*

Additional note: C11 has removed gets() (was deprecated in C99) replaced by ***gets_s()***

strl*() look better

```
strlcpy(path, homedir, sizeof(path)) ;
strlcat(path, "/", sizeof(path)) ;
strlcat(path, ".foorc", sizeof(path)) ;
len = strlen(path) ;
```

- May truncate, but no overflow

- Add checks for non testing code :

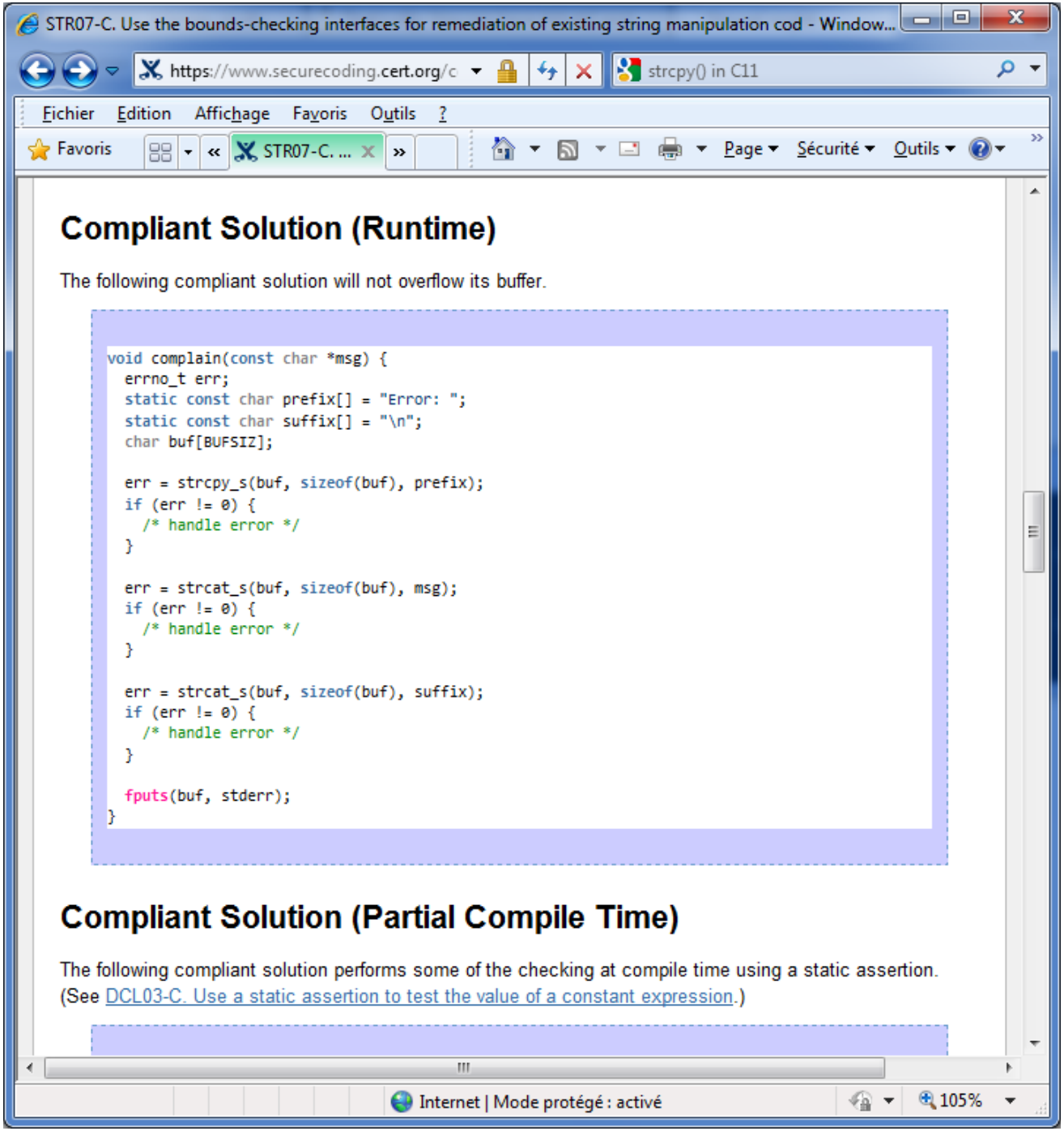
```
strlcpy(path, homedir, sizeof(path)) ;
if (len >= sizeof(path)) return (ENAMETOOLONG) ;
strlcat(path, "/", sizeof(path)) ;
if (len >= sizeof(path)) return (ENAMETOOLONG) ;
strlcat(path, ".foorc", sizeof(path)) ;
if (len >= sizeof(path)) return (ENAMETOOLONG) ;
len = strlen(path) ;
```

C11 Annex K (ISO/IEC 9899:2011)

- C11 Ann.K « Bounds-checking interfaces » defines alternative versions of standard string-handling functions (from Microsoft)
- strcpy_s(), strcat_s(), strncpy_s() and strncat_s()
- *ie* :

```
errno_t strcpy_s(
char * restrict s1,
rsize_t s1max,
const char * restrict s2
);
```

- See also : ISO/IEC TR24731-1:1999 and ISO/IEC:TR24731-2:2010 …
- Note : wchar_t



Raw C11 example
from https://www.securecoding.cert.org/

Time of check, time of use

- How to create a temp. file in /tmp without overwriting an existing file ?

```
/* Generate random file name */
name = mktemp("/tmp/tmp.XXXXXXXXXX");
/* verify file does not exist */
if (stat(name,&statbuf) == 0) {
    return EEXISTS;
}
/* ok, open it */
fd = open(name, O_RDWR);
```

- Opens a possible race condition with a concurrent process
- mktemp() deprecated in POSIX.1 (2011)

Options

- Use mkstemp() to replace both system calls

```
fd = mkstemp("/tmp/tmp.XXXXXXXXXX") ;
```

- Use O_CREAT | O_EXCL, open() flags that trigger an error if the file already exists

```
fd = open(name, O_CREAT | O_EXCL);
```

- Note the difference between fopen() and open() return types (FILE* vs. int or streams vs. file descriptors)

Arithmetic overflows

```
n = getIntFromUser();
if (n<=0 || n*sizeof(struct item) > BUFMAX){
    return EINVAL;
}
```

- If *n* is big enough, the condition will not be true
- Use :

```
n = getIntFromUser();
if (n<=0 || n > BUFMAX/sizeof(struct item)){
    return EINVAL;
}
```

Arithmetic overflows

```
n = getIntFromUser();
if (n<=0){
    return EINVAL;
}
data = (struct item *)
    malloc(n * sizeof(struct item));
if (data == NULL) {
    return ENOMEM;
}
```

- If *n* is big enough, overflow occurs and a small memory allocation is done
  - opening the path to a memory overflow
- Use calloc() !

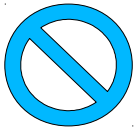
```
data = (struct item *)
    calloc(n, sizeof(struct item));
```

Format strings issues

- Many standard display functions use a format for printing : printf(), sprintf(), fprintf(), …
- Two variants exist, with and without format strings : printf("%s", ch) or printf(ch)
- What happens when you give « %x » to printf ?
  - printf() gets its next argument from the stack
- When user input is passed to such functions, it can generate this kind of situations
- This kind of situation may allow to access areas of memory for reading (sometimes for writing)

Example

```
#include <stdio.h>
int main()
{
    char * secret = "polichinelle";
    static char input[100] = {0};
    Printf("Enter your name: ");
    scanf("%s", input);
    printf("Hello ");printf(input);printf("\n");
    printf("Enter your password: ");
    scanf("%s",input);
    if (strcmp(entree,secret)==0) {
        printf("OK\n");
    } else {
        printf("Error\n");
    }
    return 0;
}
```



Example

- Normal use of the program

```
bash$ ./a.out
Enter your name: Jack
Hello Jack
Enter your password: ripper
Error
```

- « Abuse » of the program

```
bash$ ./a.out
Enter your name: %p%s
Hello 0x08049760polichinelle
```

- Allows to walk the stack and access internal program data

Practical recommendations

- Design first
  - Often broken and insecure by design
- Obscurity does not help
  - Exploits against closed source may be just as easy as against open source
  - Obfuscation primarily works for people writing code, not crackers
- Quality is security
  - Most security problems are simple bugs
  - There is no security plugin
  - No ROI for security
    - But shorter test cycles
    - Less bugs, so less time spent fixing them
    - And usually better efficiency

Practical recommendations

- Most code should be simple and boring
  - Easier to audit
  - Already formatted
  - Clever code is almost always wrong
- Fix a bug everywhere
  - Even automate for checking it
- Check return codes
- Design your APIs right...
- Understand semantics
  - File descriptors
    - Inheritance over fork
    - Access rights only checked on open()
  - Signal handlers *are* complex
    - Simple rule : only set volatile atomic flags in them

Practical recommendations

- Most security issues come from abstraction layers violation (audit these cases)
  - Hidden variables
  - Concurrency
  - Overflows
  - Flow control on error
- All user input must be checked
  - Positive checks
  - Everything not static is like user input
- Be careful with optimizations

- There is no secure language or environment
  - Java does not suffer from simple buffer overflows but has integer overflows, logic errors, etc.