Sécurité des logiciels

A few case studies

Samuel Thibault <samuel.thibault@u-bordeaux.fr>

A few case studies

- JPEG of Death (Microsoft gdiplus.dll), 2004
- Debian weak keys (openssl), 2008
- Heartbleed (openssl), 2014
- Getaddrinfo (libc), 2016
- Shellschock (bash), 2014
- Bdoc2 (Microsoft Word), 2000 (?)

JPEG of Death, 2004

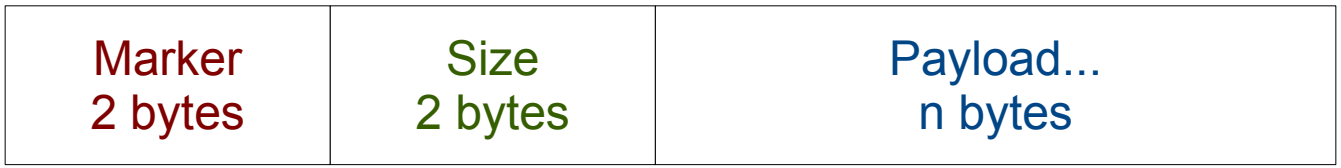
2004

- The Internet is booming
- More and more people read websites, mails, etc.
- They fetch more and more data from the Internet
  - Notably, JPEG pictures

JPEG format

A series of *segments*, each of which

- Start with a 2-byte marker, specifies the type:
  - **`FF D8`**: Start Of Image (SOI)
  - **`FF C0`**: Start Of Frame (SOF)
  - **`FF C4`**: Define Huffman tables (DHT)
  - **`FF D9`**: End of Image (EOI)
  - **`FF DA`**: Start of Scan (SOS)
  - **`FF E0`**: Application-specific 0 (APP0)
  - **`FF FE`**: Comment (COM)
  - ...
- For most of them, followed by 2-byte payload size
  - **`00 10`**: 16 bytes
- And the payload
  - ...



JPEG format

```
FF D8 FF E0 00 10 4A 46 49 46 00 01 01 00 00 01
00 01 00 00 FF C0 00 0B 08 00 DD 01 5F 01 01 11
00 FF C4 00 15 00 01 01 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 09 FF C4 00 14 10 01 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 FF DA
00 08 01 01 00 00 3F 00 [...] FF D9
```

GDI+

GDI (Graphics Device Interface)

- Shared library for drawing graphics
  - Lines, curves, text, color palettes, pictures
- Shipped since beginning of Windows

GDI+

- Extended with PNG and JPEG support
- Shipped since Windows XP / Server 2003

Applications just use them rather than embed their own picture loading support

JPEG format

- Size, but of what?
- of size + payload

GDI+ support for comments

```
unsigned int type, size, alloc_size;

type = p[i++];
type = (type << 8) + p[i++];
switch(type)....

size = p[i++];
size = (size << 8) + p[i++];

alloc_size = size + 2;
segment = malloc(alloc_size);

if (size - 2 > 0)
  memcpy(segment+4, p, size - 2);
```

GDI+ support for comments

What if e.g. size == 0 ?

- size - 2 is 2^32 - 2

- Heap overflow
- Very large heap overflow!
- Should have been crashing the program

- But SEH (Structured Exception Handler) catches this
  - Similar to **`try / except`**

GDI+ support for comments

SEH rolls back

- Assumes the data is bogus, ignores it
- Resumes with next segment
- Again, malloc(alloc_size)

But malloc management was overwritten by comment...

- I.e. malloc() uses BK / FD as address and content to write
- I.e. can target e.g. return address
- And point it at a shellcode in the comment

JPEG of the Death, in practice

- Craft a shellcode
- Embed it in a comment segment of a JPEG file with size 0
- Serve it on a webpage
- Send it by e-mail
- ...

Timeline

- 2004/09/14: Announcement
- 2004/09/22: Proof of concept
- 2004/09/23,25,27: Exploit variants
- 2004/09/28: Exploitation on AOL

Not much time to react....

Lessons learnt

- Sanitize your inputs!
  - Check, check, check
- Let it die!
  - Trying to continue after a fault looks crazy to me...

- A field that has forbidden values is paving up for trouble

Debian weak keys, 2008

DSA 1571-1

"Luciano Bello discovered that the random number generator in Debian's openssl package is predictable. [...] As a result, cryptographic key material may be guessable"

Random number generator seeded only with pid (15 bits)

- Only 98301 different random sequences!
  - 32767 on a given arch
- Between 2006/09/17 and... 2008/05/14

What happened??

Random number generator (RNG)

Computer behavior is predictable

- Usually a good thing
- No good for crypto

We need random numbers for crypto

- I mean, real random
- Not pseudo-random suite, i.e. just a suite from a seed.

More precisely, we need *entropy*

- Data that comes out of the blue
- Data that cannot be predicted
- E.g. user keypresses, mouse movement
- But also hard disk response time, network packet time, ...
- Some processors even have instructions for that

Random number generator (RNG)

A random number generator (RNG)

- Collects entropy from as many sources as possible
- Hashes it as well as possible
- Provide random data

Bug #363516

```
int ssleay_rand_bytes(unsigned char *buf, int num) {
...
  MD_Update(&m, buf, num);
  while (num > 0) {
    *(buf++) = pick_random_data(m)...;
    num--;
  }
}
```

- Returns random data
- But actually uses previous data as additional entropy

"the uninitializedness taints all the users of the openssl random number generator, producing valgrind hits throughout your program, making it unnecessarily difficult to see the wood for the trees"

Bug #363516

```
int ssleay_rand_bytes(unsigned char *buf, int num) {
...
#ifndef PURIFY
  MD_Update(&m, buf, num);
#endif
  while (num > 0) {
    *(buf++) = pick_random_data(m)...;
    num--;
  }
}
```

Already marked as optional for builds with purify

Bug #363516

```
int ssleay_rand_bytes(unsigned char *buf, int num) {
...
/* Don't add uninitialized data.
  MD_Update(&m, buf, num);
 */
  while (num > 0) {
    *(buf++) = pick_random_data(m)...;
    num--;
  }
}
```

Thus comment it out, because:

- Nice to get more entropy opportunistically
- But not nice to get valgrind false positives

Bug #363516

```
void ssleay_rand_add(const void *buf, int num, double add) {
...
/* Don't add uninitialized data.
  MD_Update(&m, buf, num);
 */
...
}
```

While at it, also do the same in another function [show vimdiff]

Except that it is the function that was supposed to **feed** the RNG with entropy…

→ No RNG feeding, only pid happens to be added on the fly!

Bug #363516

```
void ssleay_rand_add(const void *buf, int num, double add) {
...
/* Don't add uninitialized data.
  MD_Update(&m, buf, num);
 */
...
}
```

Should have been noticed:

- buf is only input (const void*)
- consequently, buf is only "used" by

```
buf = (const char*)buf + j;
```

  - Possibly detected by static analysis

Bug #363516

```
void ssleay_rand_add(const void *buf, int num, double add) {
...
/* Don't add uninitialized data.
  MD_Update(&m, buf, num);
 */
...
}
```

Should have been reviewed

- Asked for comments in Debian bug
- Asked for review on openssl-dev mailing list
  - Not actually the proper mailing list
- Should have asked openssl-team mailing list
  - Not actually mentioned in the OpenSSL documentation / web page

Bug #363516

```
void ssleay_rand_add(const void *buf, int num, double add) {
...
/* Don’t add uninitialized data.
  MD_Update(&m, buf, num);
 */
...
}
```

Should have been upstreamed

- To get more eyes, upstream CI, etc. on the changes

Bug #363516

```
void ssleay_rand_add(const void *buf, int num, double add) {
...
/* Don't add uninitialized data.
  MD_Update(&m, buf, num);
 */
...
}
```

Remained so for 20 months :/

- Producing terribly weak keys on all kinds of Debian-based systems
- Weak keys possibly transferred to e.g. embedded devices

Bug #363516

And actually...

http://www.openssl.org/docs/crypto/RAND_bytes.html at the time:
“The contents of buf is mixed into the entropy pool before retrieving the new pseudo-random bytes.”

I.e. the documentation was explicitly saying the function reads the buffer, so valgrind’s “false positives” were rather true positives.

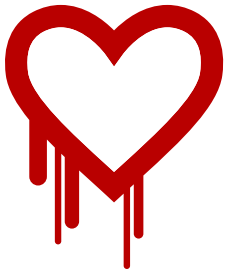
Later on (2016):
“Always DPURIFY
The use of the uninitialized buffer in the RNG has no real security benefits and is only a nuisance when using memory sanitizers.”

Lessons learnt

- Don’t blindly silence warnings
  - Fix the root cause!
  - Here, the API told explicitly that the buffer was **also** input
- Don’t keep changes for yourself
  - Nobody alone is smart enough to never make mistakes

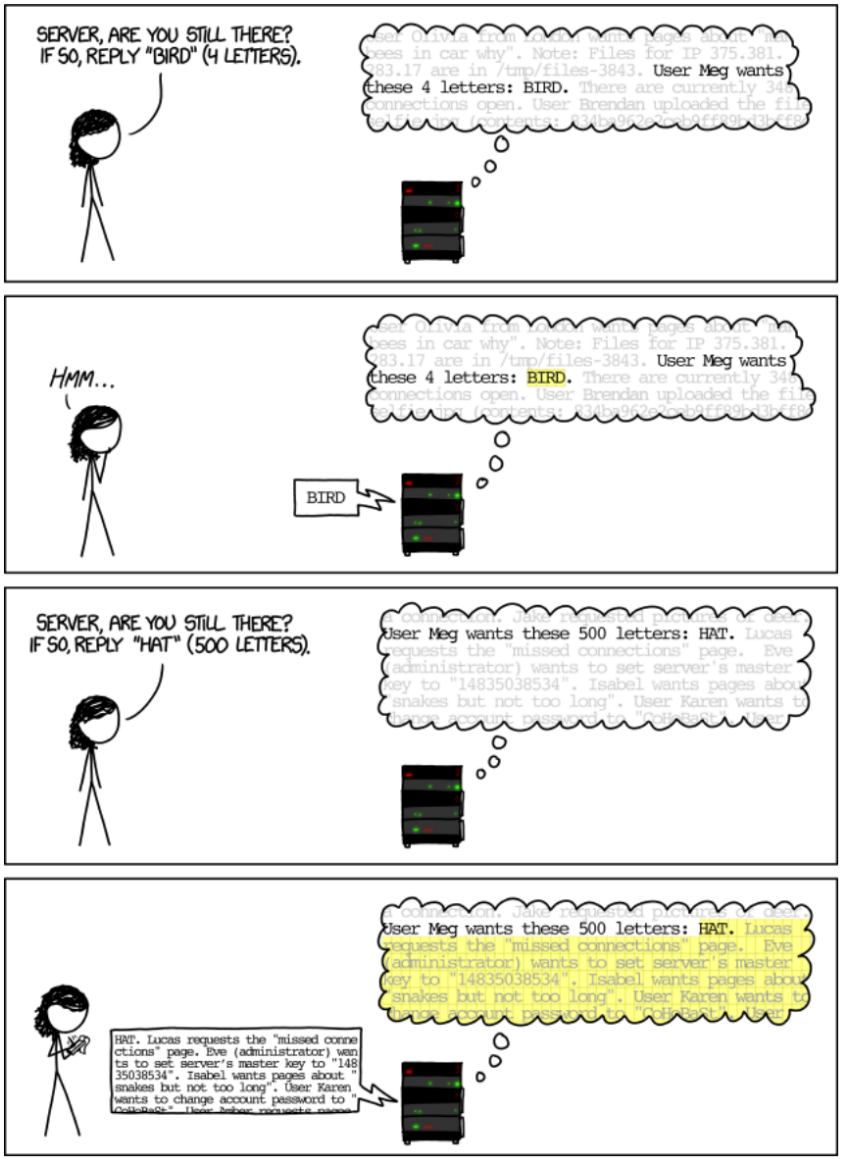
- Avoid clumsy semantic
- Make sure upstream reporting channels are properly documented

Heartbleed, 2014



Heartbleed

(xkcd 1354)



SSL/TLS

Secure Sockets Layer / Transport Layer Security

- Layer on top of TCP connections to secure them
  - https, smtps, pop3s, imaps, ...
- More details in the Network Security course ;)

Basically, exchanges records containing messages

- Ciphering negociation: key and cipher
- Encrypted data exchanges
- “Service” messages such as session key refresh

Heartbeat

Long-living TCP connection

- Usually dropped by routers on inactivity for a long period
  - Can be kept alive by getting some data through
- Possibly the other end is actually just dead
  - Need to exchange some data to make sure it's still there

But the application may not want to exchange data

- Would have to fit the application-level protocol

Exchanging data only at SSL/TLS layer?

- Previously, only way was to renegociate keys
- RFC 6520 proposes periodic "hellos" messages
  - Implemented in openssl along the RFC draft

Heartbeat

```
int dtls1_process_heartbeat(SSL *s) {
  unsigned int payload;

  hbtype = *p++;     // Packet type
  n2s(p, payload);   // Payload length
  pl = p;
  ...
  bp=buffer = malloc(1 + 2 + payload + padding);
  *bp++ = TLS1_HB_RESPONSE;
  s2n(payload, bp);
  memcpy(bp, pl, payload);
  ...
  [ send...]
```

Heartbleed

```
int dtls1_process_heartbeat(SSL *s) {
  unsigned int payload;

  hbtype = *p++;     // Packet type
  n2s(p, payload);   // Payload length
  pl = p;
  ...
  bp=buffer = malloc(1 + 2 + payload + padding);
  *bp++ = TLS1_HB_RESPONSE;
  s2n(payload, bp);
  memcpy(bp, pl, payload);
  ...
  [ send...]
```

Probably spottable by *tained* static analysis

Heartbleed

Blind memcpy

- 16bit size field: as much as 64KB
- Copies the request and anything else after it
- Most often doesn't crash: most probably 64KB worth of data

And repeatable at will

- Possibly just dumps all of the heap
- Just a question of time & luck
- Basically everything in the process compromised
  - Passwords, keys, ...

Fix: check size

[show vimdiff]

Heartbleed

Timeline

- Implementation in 2011
- Committed in OpenSSL on 31th december 2011
- RFC 6520 published in February 2012
- Release of OpenSSL 1.0.1 14th march 2012
  - With heartbeat enabled by default
- Discovery: 1st April 2014
  - 17% of trusted https servers believed to be vulnerable
- Patched: 7nd April 2014

- 20 May 2014: Still 1.5% of 800 000 most popular TLS-enabled website vulnerable
- 23 January 2017: 180 000 connected devices still vulnerable
- 11 July 2019: 90 000 connected devices still vulnerable

Lessons learnt

- Sanitize your inputs!
  - Check, check, check
- Review, review, review
  - Only one review on Hearbeat patch
  - Not enough for default-enabled security-sensitive code
- Upgrade!
  - Don’t delay security upgrades

Getaddrinfo(), 2016

getaddrinfo()

- Domain name translation
  - “www.labri.fr” → 147.210.8.54
- All Internet-related applications use it

- Ask DNS server for the translation
- Return answer to application

- Basically, whenever you type “something.com” in your browser

getaddrinfo() vulnerability

Basically

- IPv4 + IPv6 lookups in parallel
- buffer reuse, but improper buffer size recording
  → DNS answer overflowing in the heap

How to attack

- Set up a DNS server on yourdomain.com
  - That carefully crafts answers
- Point victims to http://yourdomain.com

Lessons learnt

- Playing smart with buffers is dangerous
  - Just deallocate/reallocate
  - Safety is more important than efficiency
- Code used by **all** applications
  - Very sensitive target

Shellschock, 2014

Bash is powerful

Exporting variables from a shell to a subshell

```
sh1$ A=1
sh1$ export A
sh1$ bash
sh2$ echo $A
1
sh2$ printenv
[...]
A=1
```

Bash is powerful

Exporting **functions** from a shell to a subshell

```
sh1$ f() { echo Hello; }
sh1$ export -f f
sh1$ bash
sh2$ f
Hello
sh2$ printenv
[...]
f=() { echo Hello; }
```

Very unknown feature, introduced in 1989 (bash 1.03)

Bash is **too** powerful

Exporting **functions** from a shell to a subshell

```
sh1$ f='() { echo Hello; }'
sh1$ export f
sh1$ bash
sh2$ f
Hello
sh2$ printenv
[...]
f=() { echo Hello; }
```

Bash is also bugged

Exporting **functions** from a shell to a subshell

```
sh1$ f='() { echo Hello; }; echo Heya'
sh1$ export f
sh1$ bash
Heya
sh2$ f
Hello
sh2$ printenv
[...]
f=() { echo Hello; }; echo Heya
```

Bug apparently introduced in 1992

What does it mean

Terribly powerful vulnerability

- Setting an environment variable is enough
- Any variable name is fine
- Just needs to control the content
  - `f='() { :; }; whatever you want'`
- And execute a shell
  - E.g. through system(), popen(), ...
- "**`whatever you want`**" will be executed
  - Even if **`f`** is not called!

What does it mean

Environment variables set from tainted input

- Dynamic websites (php/asp/etc.)
  - QUERY_STRING
  - HTTP_USER_AGENT
  - HTTP_COOKIE
- DHCP service
  - new_domain_name
- Mail service
  - Program-based mail delivery
- ...

Bug fixed

```
sh1$ f='() { echo Hello; }; echo foo'
sh1$ export f
sh1$ bash
bash: warning: f: ignoring function definition
attempt
bash: error importing function definition for `f'
sh2$ f
bash: f: command not found
sh2$ printenv
[...]
f=() { echo Hello; }; echo foo
```

In... 2014, 22 years (!) after supposed bug introduction

Bug fixed

```
sh1$ f='() { echo Hello; }'
sh1$ export f
sh1$ bash
sh2$ f
Hello
sh2$ printenv
[...]
f=() { echo Hello; }
```

Still scary

Bug fixed, functionality hardened

```
sh1$ f() { echo Hello; }
sh1$ export -f f
sh1$ bash
sh2$ f
Hello
sh2$ printenv
[...]
BASH_FUNC_f%%=() { echo Hello; }
```

Normally never crafted by anything else but bash

Lessons learnt

Did I already say “sanitize your inputs”? :)

Basically unknown feature

- Nobody thinks about checking security there

Powerful feature

- Unexpected consequences

Bdoc2, 2000 (?)

永远的祝福 ，生日快乐 !!!

2000

Libraries have computers

- Write documents, save on a floppy disk (yes: 2000)
- Re-open at home, work on it, save on a floppy disk
- Re-open at library
- etc.

Not necessarily Internet, but already an infection vector

Microsoft Word

Dreadful document macros

- Documents contain code!
- Allows document processing automation
  - Very convenient!

Problem:

- AutoOpen()
- AutoClose()
- AutoExec()
- AutoExit()

Microsoft Word

[show virus.txt]

- AutoOpen()
  - Copy itself to the normal template and active document
- AutoClose()
  - Copy itself to the normal template and active document
  - On September 2nd, print a message
- AutoExec()
  - Disable menu for disabling macros
- AutoExit()
  - On 13 of the month, print a message

Lessons learnt

- Any information transmission vector can be infection vector

- Powerful features make for powerful viruses
- Auto-* considered harmful
  - Dreadful autorun

Just for fun

```
ILOVEGNU
I am the "ILOVEGNU" signature virus. Just copy me
to your signature.
This email was infected under the terms of the
GNU General Public License.
```

Conclusion

Sanitize your inputs!

Shared components are high security targets

- Large attack surface through applications using them
- Well-known behavior

Powerful features

- Powerful vulnerabilities