Software Development Security

Marc ESPIE

19 avril 2018

Table des matières

1 BASIC Security

One can get the slides at https://www.lse.epita.fr/teaching/courses.html

1.1 Introduction

1.1.1 Perpectives

There are more conferences for attackers than for safety.

Perspectives :
- basic ecosystem of software from security perspective
- vocabulary to pass intership
- Dispell misconceptions about dev security

Setting limits :
- C and Unix
- All the bases of programs are in C or assembly languages

Unix is easier to start with, it's well known and well defined it has existed for more than 20 years. Then it will be not that hard to go on other OSs.

BOOKS :
- Building Secure Software(Viega, McGraw, ISBN 0-201-72152-X)
- Open BSD papers

1.1.2 Coming Exam

Simple questions for the exam :
- You will have access to lecture notes
- So if I ask you to define a term, you shouldn't copy your notes
- You should be able demonstrate that yoou understand the term by explaining it your own words - Give concrete examples
- Create your own examples

Adanced Questions :
- There will be source code and sample audits
- It won't be 100% clean
- It won't be exactly like 'epita standard code'
- If it's different it may not be wrong
- Beware of wrong asssumptions
- The security issues to fix will be nasty ones

Exam in general :
- you can write wether in french or english

1.2 Classical companies

1.2.1 Classical way companies write specs

One person by task

But it's not a good idea.

1.2.2 Auditors

If an auditor finds a bug
...Sometime it's because the design is wrong
Auditors can't catch all
... So devs must know about good practices

You can't have only one developer to remove all the security breaches it's too much.

1.2.3 Testers

You can't have pit testers VS devellopers. A good tester is invaluable.

1.2.4 Sadly

A lot of good databases experts don't even know about SQL injections.

1.3 For a release

1.3.1 Mozilla

When Mozilla ship a new version of firefox, they give the code.
But they have to adapt it for each OS.
So they do modify it before.

So after the realease, they are branchings.
Some people have to still work on correcting bugs.

When working on bugs, you can produce more bugs.

At some point you have to say that some versions are no more supported.

1.3.2 Life of a product

EOL End of life for a product
ESR Extended support release

For Windows XP, microsoft had to support for a very long time this version, because companies needed to use it and didn't wanted to change it.

Companies prefer to have a stable version of a program with security breach a program less stable but with security corrected.

1.4 Bugs

1.4.1 Security and bugs

A bug is not a 'security Hole'.
Most attacks are a serie of bugs.
We want to have defense in depth.
Fixing one bug stops the attack !
An attack is also called an exploit.
Software has vulnerabilities.

1.4.2 Finding a bug

- A developer can find a bug.
- External user find the bug and report it to the company.
- Others can sell this bug.

1.4.3 What to do

Be proactive about a security issue.
Fixing it without letting the bad guys know.

A soon as you find something, you have to fix it, because there will be an soon or later.

1.4.4 Little history

25 years ago, there was a mailing list called bugtraq, but multiple times you had the same new on different OSs.
Now there is CVE
CVE : common vulnerabilities and exposures.

1.4.5 Reporting a bug

Don't do it on Friday
Account for vendors
Have a "secure" channel for bugs

"Meltdown" for Intel processors : is a good example of not what to do.

1.4.6 Worst case scenario : zero days

It's when you see people exploiting a bug.

1.4.7 Bad behaviors

- It's too complicated to be exploitable
It will take time but it will be exploited.

The IE5 url overflow :
Use it to craft assembly code to craft more overflow.
Two month later, a guy used it to write self modifying code.

1.4.8 Be prepared

Sofware components get reused all the time.
Plan to be successfull.

Your code may be used in Hospitals, so nasty guys could exploit it and could kill people.

1.4.9 Closed source

It's no more secure.
lot's of people know how to reverse engineer.

You want to avoid the 'sweep under the carpet effect'

If you do open source, some people will look at what you are doing.

example :
A guy recovered most patterns of buffer overflow from microsoft windows.
It takes one bug...
Everything is exploitable.
You have to carefully check the code that you have been writting.

1.4.10 How to avoid bugs

- Don't do bugs
- Know you APIs
- Prefer secure idioms

1.5 Avoiding bug exploit

1.5.1 Mitigation

In OS, it's an actual technique that will mitigate it.
example : canaries
Function prolog will insert random data on the stack.
This will be checked at the end of the function, if it has changed then, stack smash if called to make program end.

1.5.2 Guard pages

Pages are separed in order to avoid writting on other pages.

1.5.3 Better API's

Don't use strcpy, strcat, strncpy nor strncat.
prefer strlcpy, strlcat ; so you can use the size to see if it will fit.

1.5.4 The Drepper fallacy

- 'But I don't write wrong code'
- The reason for slow adoption of strlcpy
For over 10 year Mr Drepper didn't wanted to introduce strlcpy on linux, because people had to write good stuff.
Prefer snprint to sprintf.
When writing code in C, the size have to be obvious.
90% of all software is :
- crap
- unimportant to optimize
- etc...
You can't fix everything
... therefore don't fix everything

This produces "Low-Hanging fruits"
As long as you avoid basic bugs, you're safe from most pirates. (95% of them)

2 Unsecure programs

2.1 Buffer Overflow

Be carefull with malloc overflows.
The programm opening an image can be vulnerable to overflows.
ie : flash with buffer overflows. If you did the right overflow, you could have get more rights than you should have had.
You have to write correctly your library so it is fast and safe.
Beware of undefined behaviors and compiler optimizations.
When working with arrays in C, you want to check the size.
Interger overflow => undefined behavior
You have to check th size of max integer depending on your CPU.

Broken code :
Good practices :
— Use calloc. Make sure that it is safe.
— Maybe craft you own.

2.2 SQL injection

In SQL, never use do.
Don't assume the people you work with know about this.
Never do matching about negative patterns.
ie : Some patterns can be added later, and so you won't be protected anymore.

You should accept the less things you can, and open more things when people ask you to do it. In order to avoid security leaks.

2.3 More about quoting

What you don't know will kill you
Never do matching against negative patterns
e.g., an email address is not something that does not contain some characters
Is is something that only matches a given pattern.
(Subsidiary question : figure out a regexp that matches email)

Do positive matching.

2.4 Printf

What you don't know will kill you.
Printf can print part of memories that you don't want to share.
Beware of executing shell commands from user or asking for paths.

Always write printf("%s", msg);
The compiler will optimize it.

2.5 Forgotten shells

Every time one launches system or popen, a shell is run.

2.6 Services

When a service crash, what do you do ?
-restart service
-don't restart it

The good idea is not to restart it yet, and look at the logs.
So you can know where is the problem.
There may be attackers.

If you have a system that gives you notifications or warning messages, you have to use them.
You should try to attack your owns servers, and make them fail, in order to be sure they fail for the good reasons.
Make sure that you verify regularly that things can fail.
Netflix had some of the same issues, it was client of AWS.
https://www.lemondeinformatique.fr/actualites/lire-netflix-libere-chaos-monkey-dans-la-jungle-open-
html