

Développement sécurisé dédié aux applications embarquées

Xavier Kauffmann

Xavier Kauffmann

- Ingénieur, Docteur, Hacker / Bug Hunter :: Sécurité Embarquée
- Security Architect chez [SumUp](#)
- Expert sécurité chez [Ingenico](#)
- [Thèse CIFRE](#) chez [Oberthur Technologies](#)
- Ancien STI promo 2008
- [@email](#)



Vous?

- Matières
- Projets
- Ce que vous voulez faire?



“ Sécurité embarquée:
reverse engineering, analyse de code, crypto, side channel,
lasers, pentesting ...”

Cours



TD



Plan

QUI? Contre qui on se bat?

QUOI? Règles du jeu?

COMMENT? Les soluces

.

Contre qui on se bat?

- Profiler un attaquant
- Surface d'attaque?



Qui sont les joueurs?

- Acteurs, leurs forces, leurs faiblesses
- Entreprises / universités / labos / certifications
- Sécurité en entreprise: un travail d'équipe
- Comprendre les objectifs
- Où est ce que vous intervenez? votre place?



Quelles sont les règles du jeu?

- Point commun avec le développement standard
- Spécificités de l'embarqué
- Les attaques / les défenses



Ne pas s'arrêter à la surface



Regarder sous le capot



Niveaux d'abstraction

Software → Hardware

- Architecture
- Flot information
- Couverture défensive
- Process

Sécurité checklist

- Produit / Service?
- Threat modeling
- Reverse engineering / Pentesting
- Exploit
- Dev / Defense
- Security architecture
- Process / compliance

Les soluces

- Comprendre les niveaux d'abstraction
- Défense et architecture
- Conscience faiblesse de l'armure

- Appréhender les outils
- Qualités à avoir / amélioration continue
- Identifier plus value (temps, \$\$\$, charge)

Skillz

Reverse engineering / protection

Attaques / défenses compilation

Sécurité au niveau projet



POWER LEVELING

When you just don't have the time

Reverse engineering / protection

- Analyse de binaire / patching
- Bootloader et **chaine de confiance**
- Protections: obfuscation / signature / crypto

Comprendre ce que peut faire l'attaquant

Comprendre la sécurité à l'échelle d'un produit

Attaques / défenses compilation

- Attaques software et hardware
- Défenses associées
- Les bugs ne sont pas vos amis
- Use case

Aller au contact du code

Comprendre et empiler de la défense

Sécurité d'un projet

→ Incorporer la sécurité dans un projet

- Analyse statique
- Fuzzing
- Couverture de code

Mettre en place des bonnes pratiques
Implémenter des tests de sécurité

1 - Contre **QUI** on se bat?

Sécurité checklist

- **Produit / Service?**
- **Threat modeling**
- Reverse engineering / Pentesting
- Exploit
- Dev / Defense
- Security architecture
- Process / compliance

Embedded Systems



Le trésor

Les assets sensibles

- * Clés crypto
- * Données utilisateurs
- * Droit d'accès
- * Fonctionnel



Profiling ze attacker

Modèle d'attaque

- Différents profils
- Différentes cibles: iphone / console / ...
- Moyen infiltrer / exfiltrer des informations
- Cherche persistance



Les motivations

→ ze gold!

mais pas que ... ←

.



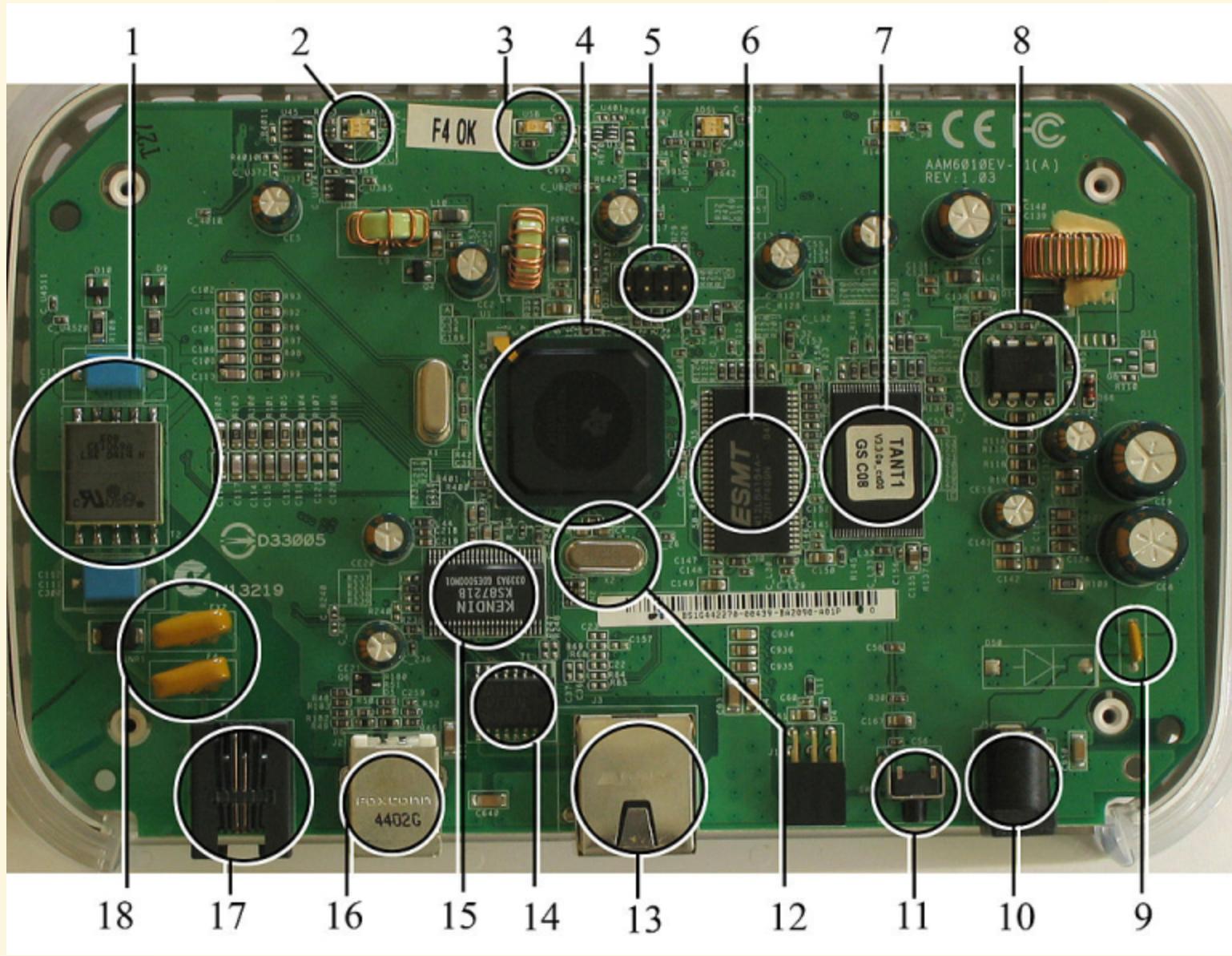
Attackers connus



[Geohot | Tarnovsky](#)

Analyse de sécurité

- Déterminer les **points d'entrées** / d'attaques
- Corrélation **asset / défense**
 - Quels sont les points de contrôles disponibles ?
 - Quelles sont les attaques connues dans ce contexte ?
- Vision **macroscopique / microscopique**
- Où intervient la sécurité dans le SDC ? Quel est la durée de vie de mon produit ?
- Finalement plonger dans **code**



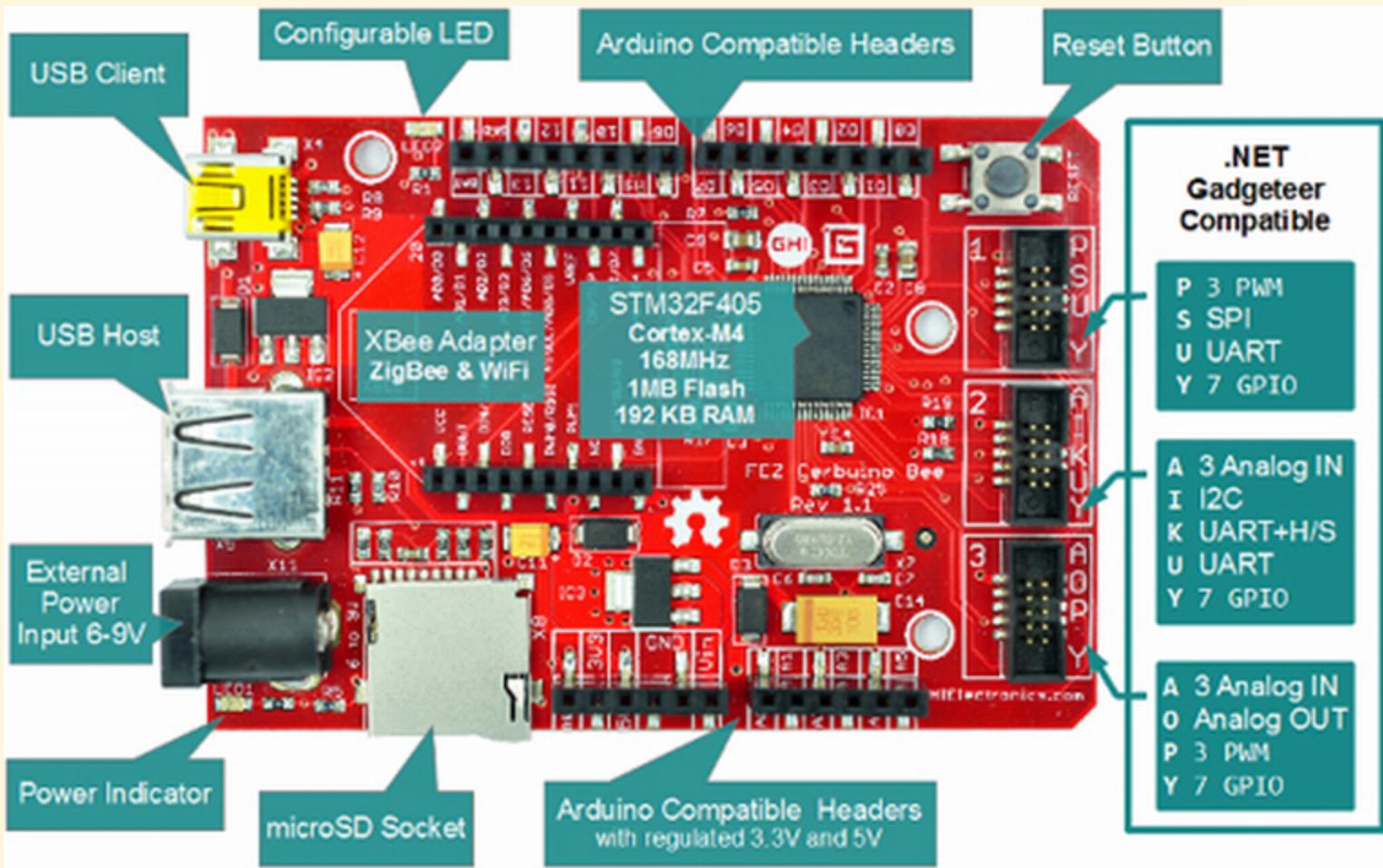
Surface d'attaque et points d'entrée

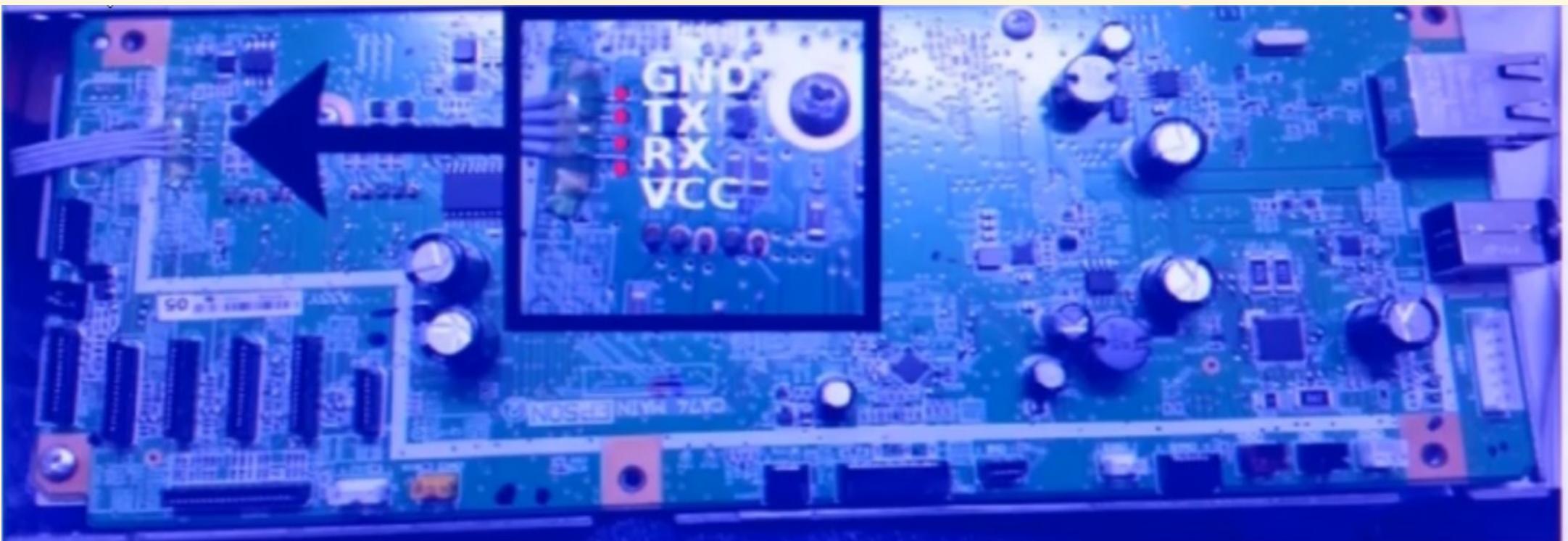
1. Telephone decoupling electronics (for ADSL)
2. Multicolour LED (displaying network status)
3. Single colour LED (displaying USB status)
4. Main processor, a TNETD7300GDU
5. JTAG test and programming port
6. RAM, a single ESMT M12L64164A 8 MB chip
7. Flash memory, obscured by sticker
8. Power supply regulator
9. Main power supply fuse
10. Power connector
11. Reset button
12. Quartz crystal
13. Ethernet port
14. Ethernet transformer, Delta LF8505
15. KS8721B ethernet PHY transmitter receiver
16. USB port
17. Telephone (RJ11) port
18. Telephone connector fuses

Spécificité embarquée

- **Hardware**
 - Connectivité: uart, rs232, spi, i2c, telnet, ssh,
- **Software**
 - Environnement constraint, architecture, jeu d'instruction

Standard	Tx Type	# Signal Wires	Data Rate & Distance	Hardware \$	Scalability	Application Example
UART	Asynchronous	2	20kbps @ 15m	Medium (transceiver)	Low (point-to-point)	Diagnostic display
LIN	Asynchronous	2	20kbps @ 40m	Medium (transceiver)	High (identifier)	Washing machine subsystem network
SPI	Synchronous	4+	25Mbps @ 0.1m	Low	Medium (chip selects)	High speed chip to chip link
I2C	Synchronous	2	1Mbps @ 0.5m	Low (resistors)	High (identifier)	System sensor network





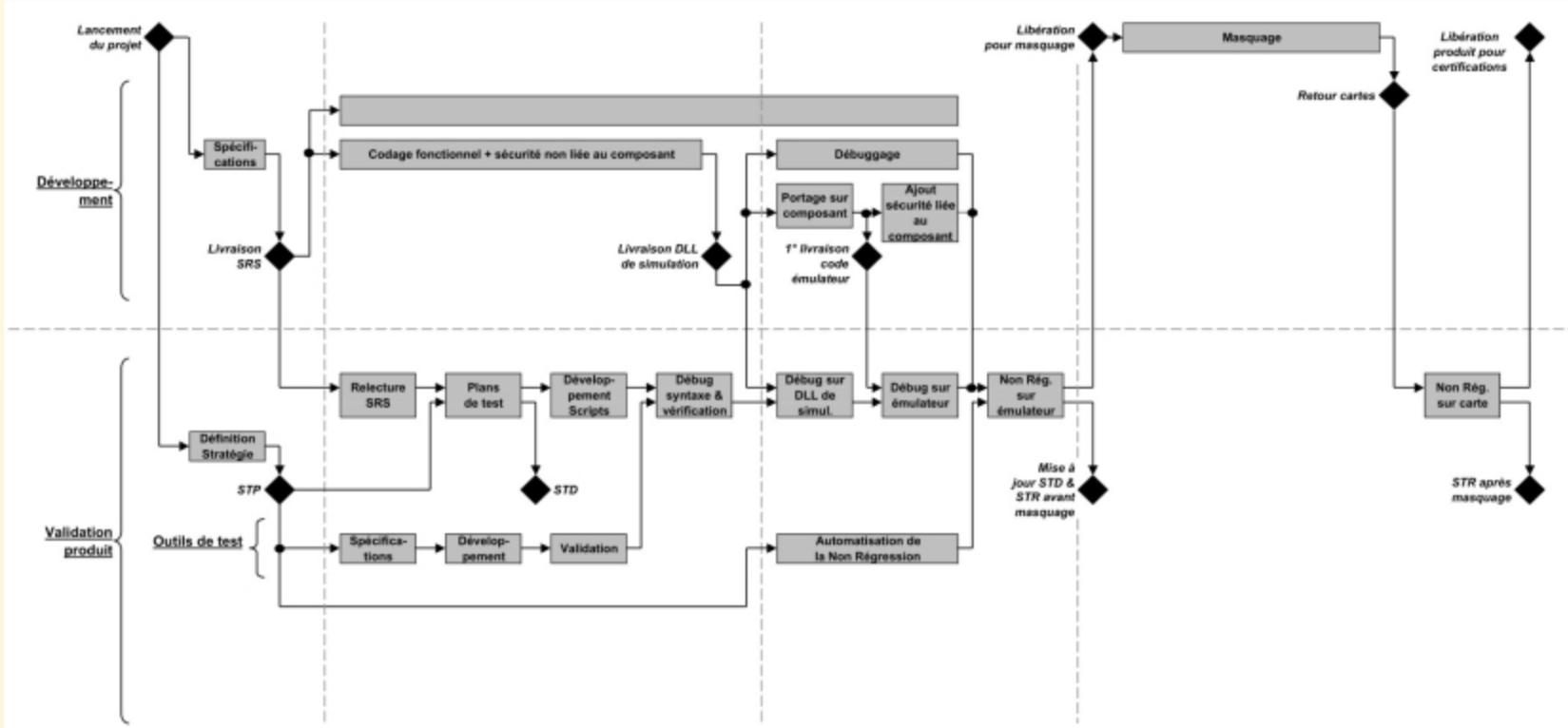
Jeu d'instructions différents

```
C:\Keil\C51\INC\REG51.H
34
35
36 /* BIT Register */
37 /* PSW */
38 sbit CY = 0xB7;
39 sbit AC = 0xB6;
40 sbit FO = 0xB5;
41 sbit RS1 = 0xB4;
42 sbit RSO = 0xB3;
43 sbit OV = 0xB2;
44 sbit P = 0xB0;
45
46 /* TCON */
47 sbit TF1 = 0x0F;
48 sbit TD1 = 0x0E;
49 sbit TF0 = 0x0D;
50 sbit TD0 = 0x0C;
51 sbit IE1 = 0x0B;
52 sbit IT1 = 0x0A;
53 sbit IRO = 0x09;
54 sbit ITO = 0x08;
55
56 /* IE */
57 sbit EA = 0xAF;
58 sbit ES = 0xAC;
59 sbit ET1 = 0xAB;
60 sbit EX1 = 0xAA;
61 sbit ET0 = 0xA9;
62 sbit EX0 = 0xA8;
63
64 /* IP */
65 sbit PI = 0xBC;
66 sbit PT1 = 0xBB;
67 sbit PX1 = 0xBA;
68 sbit PT0 = 0xB9;
69 sbit PX0 = 0xB8;
70
71 /* P3 */
72 sbit RD = 0xB7;
73 sbit WR = 0xB6;
74 sbit TI = 0xB5;
75 sbit TO = 0xB4;
76 sbit INT1 = 0xB3;
77 sbit INTO = 0xB2;
78 sbit TXD = 0xB1;
79 sbit RXD = 0xB0;
80
81 /* SCON */
82 sbit SMO = 0x9E;
83 sbit SMI = 0x9E;

C:\Keil\C51\INC\REG52.H
41
42 /* BIT Registers */
43 /* PSW */
44 sbit CY = PSW^7;
45 sbit AC = PSW^6;
46 sbit FO = PSW^5;
47 sbit RS1 = PSW^4;
48 sbit RSO = PSW^3;
49 sbit OV = PSW^2;
50 sbit P = PSW^0; //0052 only
51
52 /* TCON */
53 sbit TF1 = TCON^7;
54 sbit TD1 = TCON^6;
55 sbit TF0 = TCON^5;
56 sbit TD0 = TCON^4;
57 sbit IE1 = TCON^3;
58 sbit IT1 = TCON^2;
59 sbit IRO = TCON^1;
60 sbit ITO = TCON^0;
61
62 /* IE */
63 sbit EA = IE^7;
64 sbit ES = IE^5; //0052 only
65 sbit ET1 = IE^4;
66 sbit EX1 = IE^3;
67 sbit ET0 = IE^2;
68 sbit EX0 = IE^1;
69 sbit RXD = IE^0;
70
71 /* IP */
72 sbit PT2 = IP^5;
73 sbit DS = IP^4;
74 sbit PI = IP^3;
75 sbit PX1 = IP^2;
76 sbit PT0 = IP^1;
77 sbit PX0 = IP^0;
78
79 /* P3 */
80 sbit RD = P3^7;
81 sbit WR = P3^6;
82 sbit TI = P3^5;
83 sbit TO = P3^4;
84 sbit INT1 = P3^3;
85 sbit INTO = P3^2;
86 sbit TXD = P3^1;
87 sbit RXD = P3^0;
88
89 /* SCON */
90 /* SCON */
```

Analyse de sécurité (suite)

- Cycle de vie d'une carte: fabrication, initialisation, personnalisation, utilisation, mort
- Corrélation asset / **propriétés de sécurité**
- Cycle de développement



Sécurité checklist

- *Produit + Surface d'attaque*
- *Qui nous attaque + Pourquoi*
- **Reverse engineering / Pentesting**
- **Exploit**
- Dev / Defense
- Security architecture
- Process / compliance

Pentesting

- Integration en entreprise / équipe
- **Compétences**
 - Reverse engineering / Forensics
 - Threat modeling
 - Code review / Vulnerability development
 - Crypto

WARNING:

I
H
A
V
E
M
A
D
N
I
N
J
A
S
K
I
L
L
S



Qu'est ce que je peux OBSERVER?

Qu'est ce que je peux MODIFIER?

Entreprise

- Stratégie
- Vitesse
- Taille
- Coût



Certification

- Pourquoi certification?
- Pourquoi labo de sécurité?
- Procédure audit, cotation attaque: accès / dégats

anssi fips cspn
pci-dss cissp
critères-communs
cehpcipci-pts

Résumons

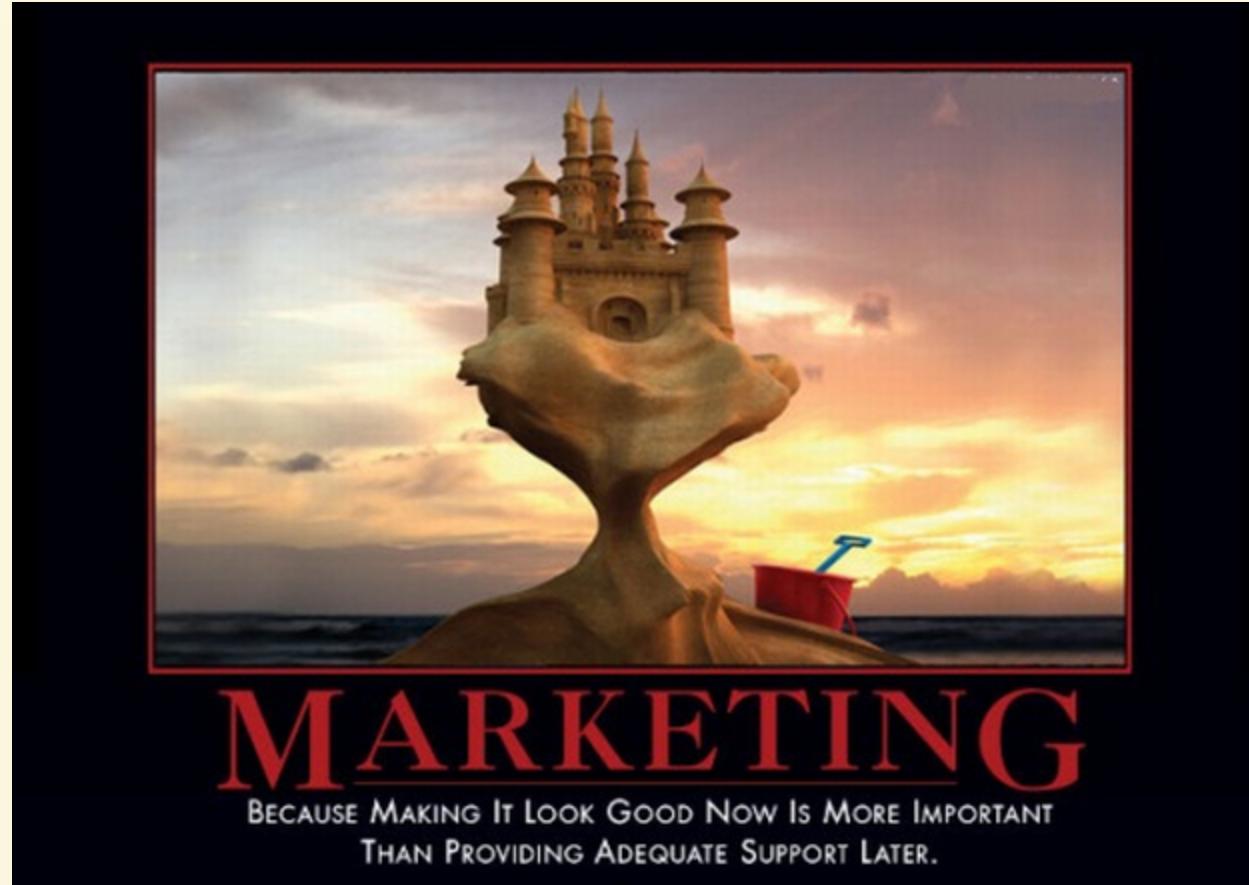
“ Commencer par modéliser l'attaquant par rapport à ce qu'on veut protéger

Coller cette représentation sur le produit et son architecture

Niveaux d'abstraction

Enfin introduire et contrôler la sécurité dans le cycle de développement ”

Se méfier de ...



Sécurité checklist

- *Produit + Surface d'attaque*
- *Qui nous attaque + Pourquoi*
- **Reverse engineering / Pentesting**
- **Exploit ➔ TD!**
- Dev / Defense
- Security architecture
- Process / compliance

TD time!

TD1



2 -Les règles c'est **QUOI** ?

Sécurité checklist

- *Produit + Surface d'attaque*
- *Qui nous attaque + Pourquoi*
- *Risque de sécurité*
- *Menace / Attaque*
- **Dev / Defense**
- **Security architecture**
- *Process / compliance*

Regarder sous le capot



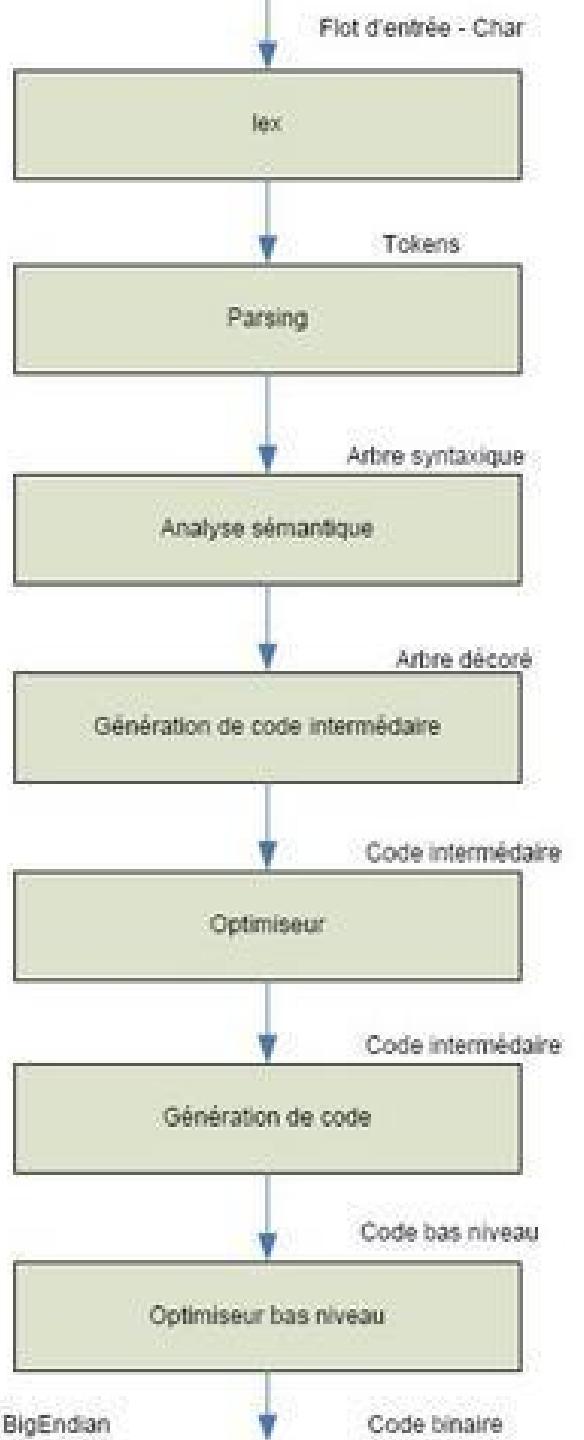
“ 20 000 lieux sous les mers
Du code au silicium ”

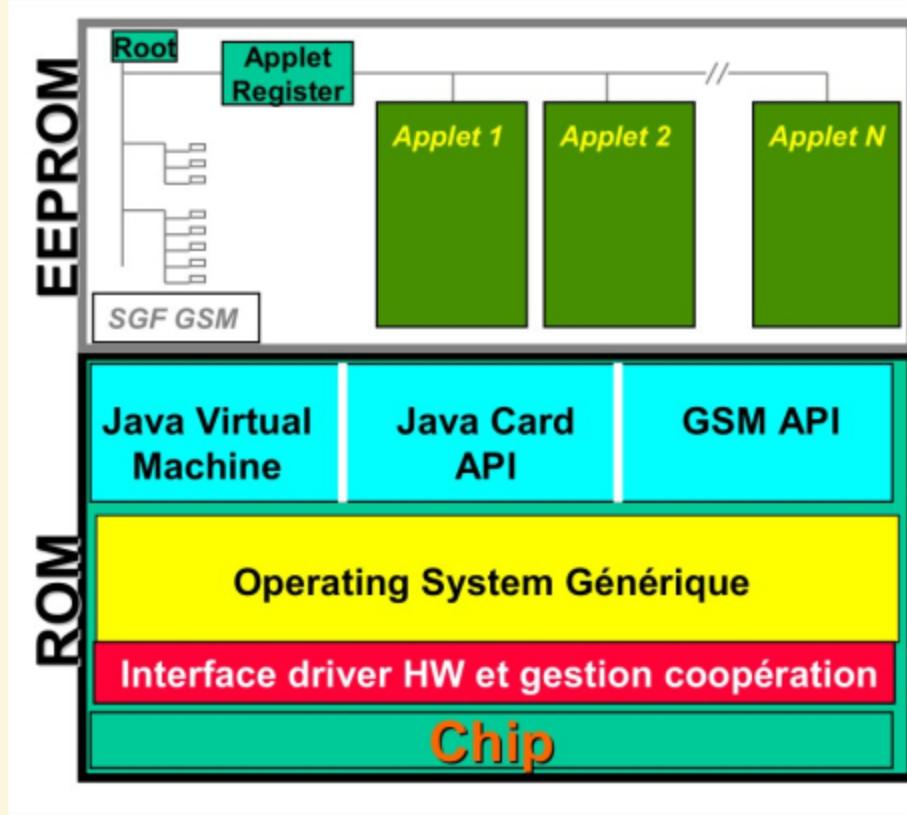
Il était une fois le code ...

```
#include <stdio.h>
main() {
    printf("Hello world");
}
```

Dans un système très très
lointain ...







PC: Program Counter

- lit le code binaire
- l'instruction codée est exécutée par le microprocesseur

011**01011**01011101110110101001

Instructions
Condition
Loop
Jump

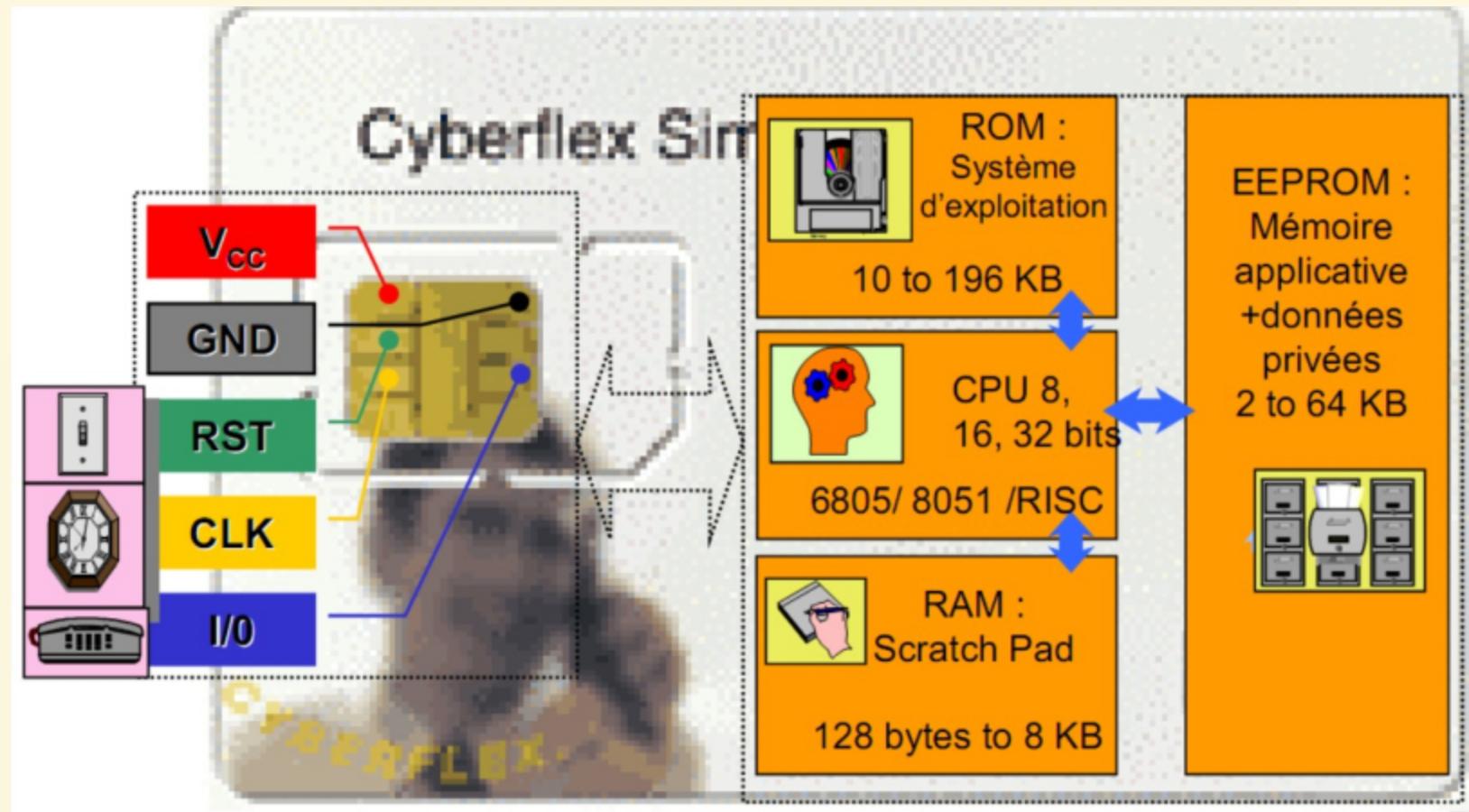
0x00

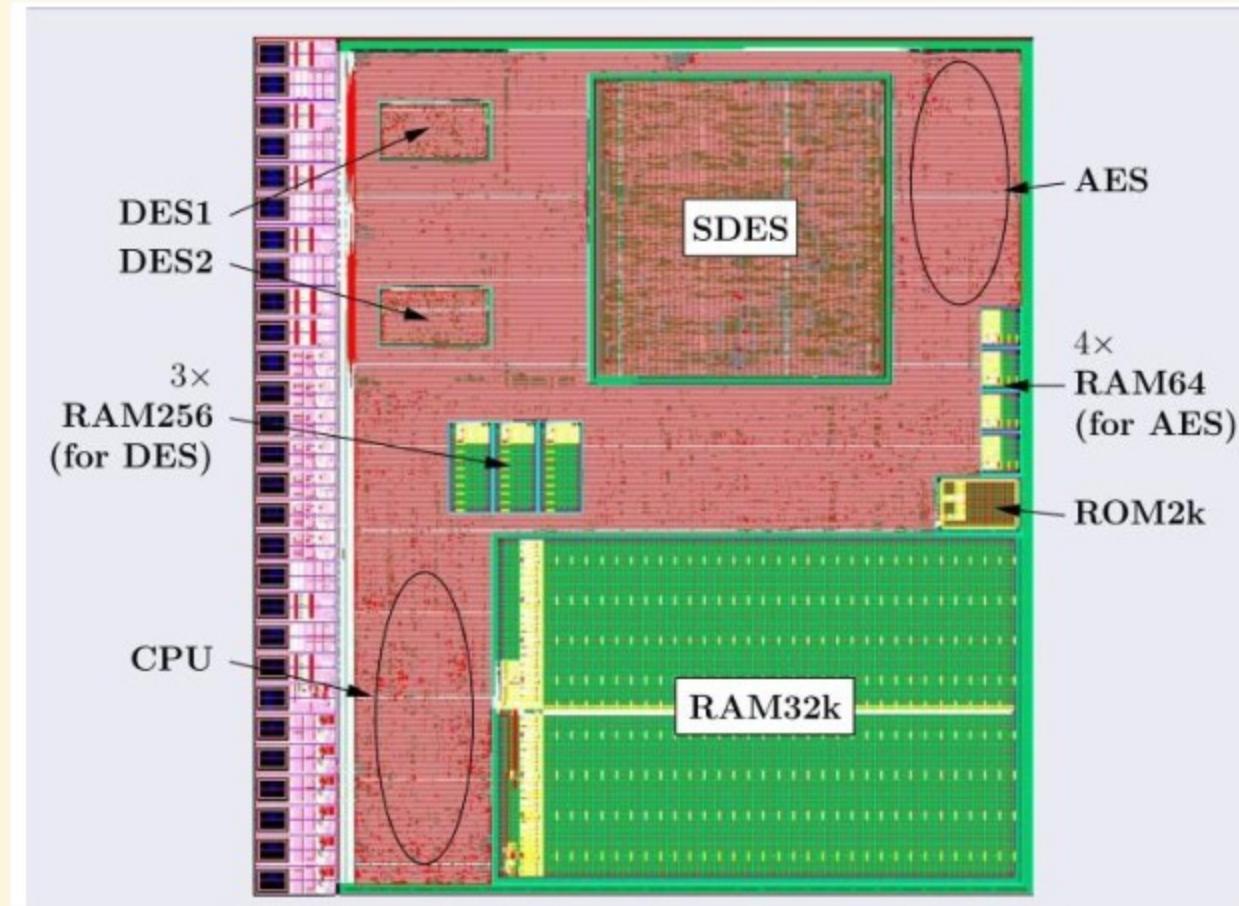


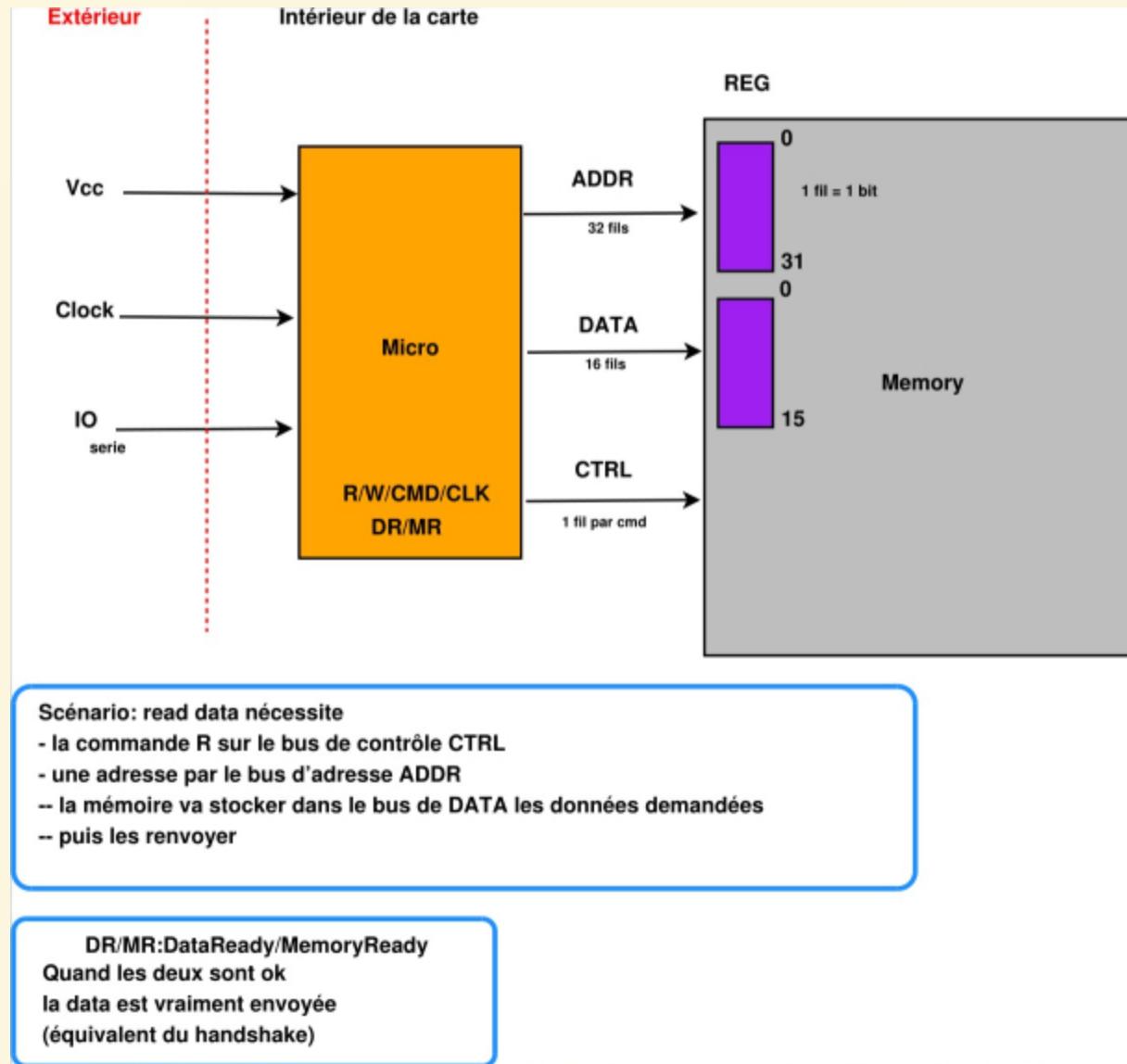
0xFF

SP: Stack Pointer

- navigue dans la pile d'adresse
- change/restaure le contexte d'exécution lors de saut dans le code



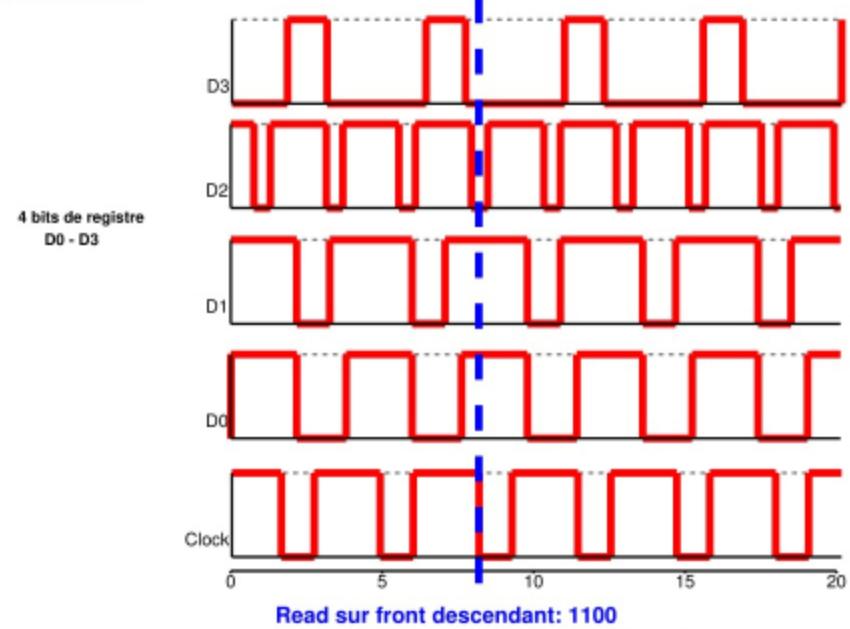




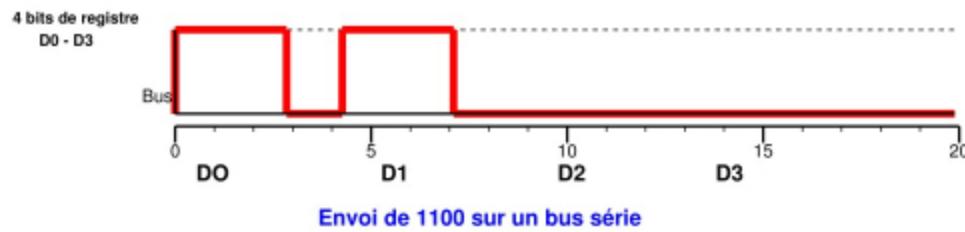
Energie et temps



Bus Parallèle



Bus Série

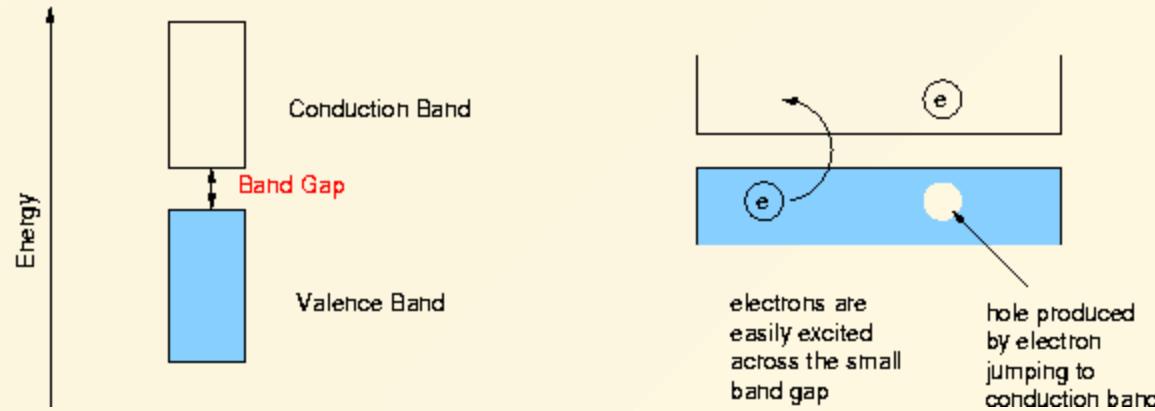


Envoi de 1100 sur un bus série

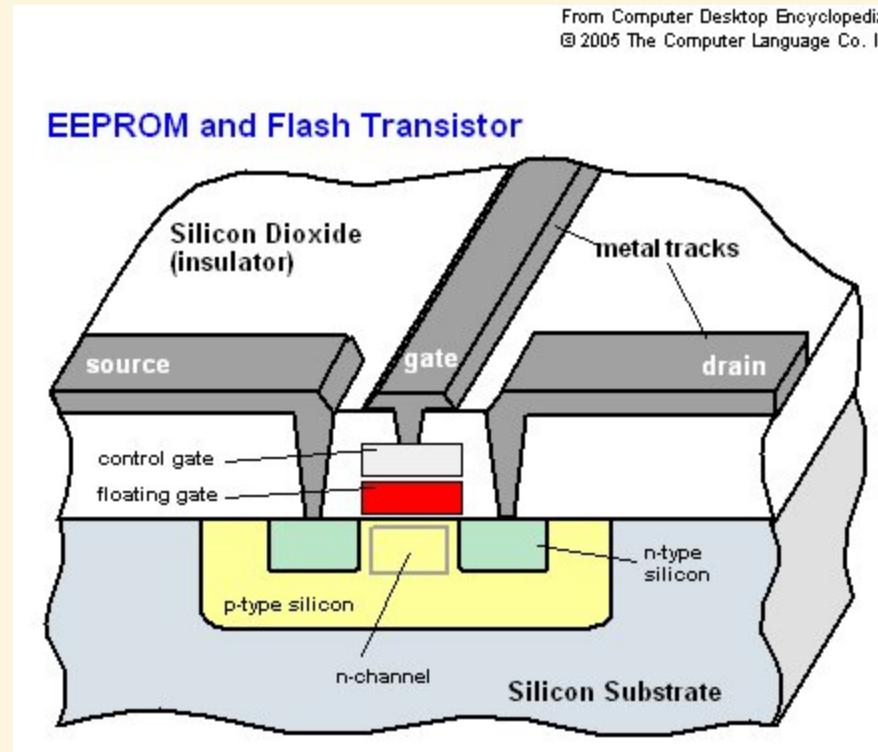
Cablé dans le composant:

- *génération clock / timers
- *communication hardcodé de la mémoire interne
 - correction d'erreur
 - retry
- *interruption hard

Porte logique



Mémoire



Minecraft Beta 1.8.1
flying



Recette

- Energie x1
 - Temps x1
 - Logique x1
-
- Silicium x1
 - Transistor x1

```
if(err != 0)
    printf("Errorcode=%i\n", err);
else
    printf("OK!\n");
```

```
CMP r1, #0
BEQ .L4
LDR r0, <string_1_address>
BL printf
B .L8
.L4:
LDR r0, <string_2_address>
BL printf
.L8:
```

Assembleur Land

“ Un monde d'affectations mémoire et de sauts

,,



Sécurité checklist

- *Produit + Surface d'attaque*
- *Qui nous attaque + Pourquoi*
- *Risque de sécurité*
- *Menace / Attaque*
- **Dev / Defense**
- **Security architecture**
- *Process / compliance*

Sécurité checklist

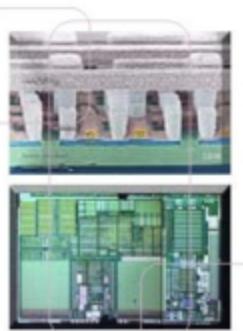
- *Produit + Surface d'attaque*
- *Qui nous attaque + Pourquoi*
- *Risque de sécurité*
- *Menace / Attaque*
- **Comprendre les règles du système**
- **Attaques?**
- Process / compliance

Attaques

Over 25 years of race between designers and hackers on HW/SW protection

● Invasive attacks

- RNG corruption
- Bus probing
- Focus Ion Beam
- Pico-second Imaging
- ...



● Observation attacks

- Timing
- DPA, SPA
- Electromagnetic
- Alpha-particles
- ...



■ SW attacks

- Trojan horse
- Differential Fault analysis
- Protocols
- Cryptography



■ Side channel attacks

- Glitches injection
- Light attacks
- Laser pulses
- ...



■ Independent certification

- FIPS
- Common Criteria



```
if(err != 0)
    printf("Errorcode=%i\n", err);
else
    printf("OK!\n");
```

```
CMP r1, #0
BEQ .L4
LDR r0, <string_1_address>
BL printf
~~B .L8~~ /* -glitch!- */
.L4:
LDR r0, <string_2_address>
BL printf
.L8:
```

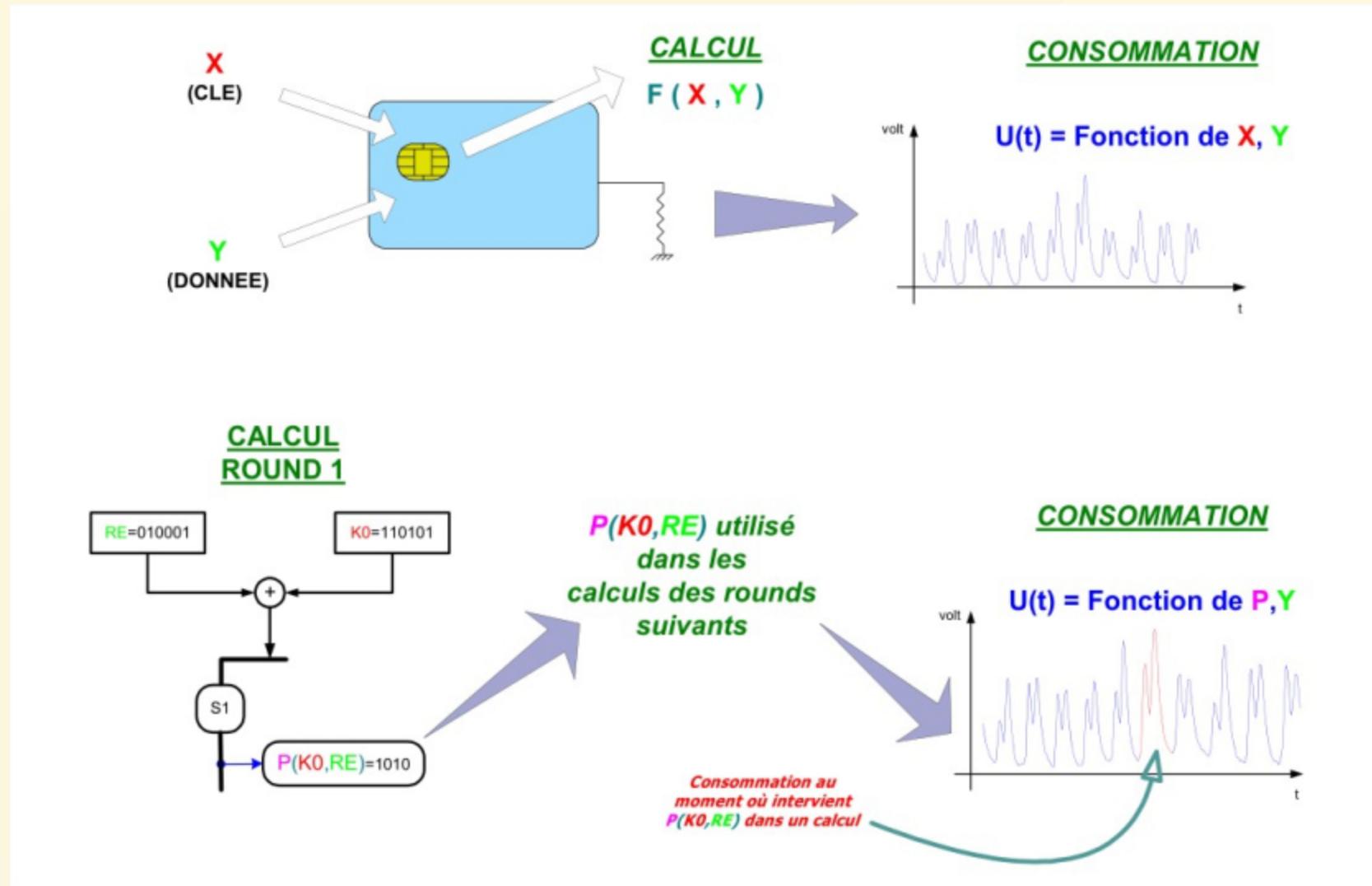
Une solution

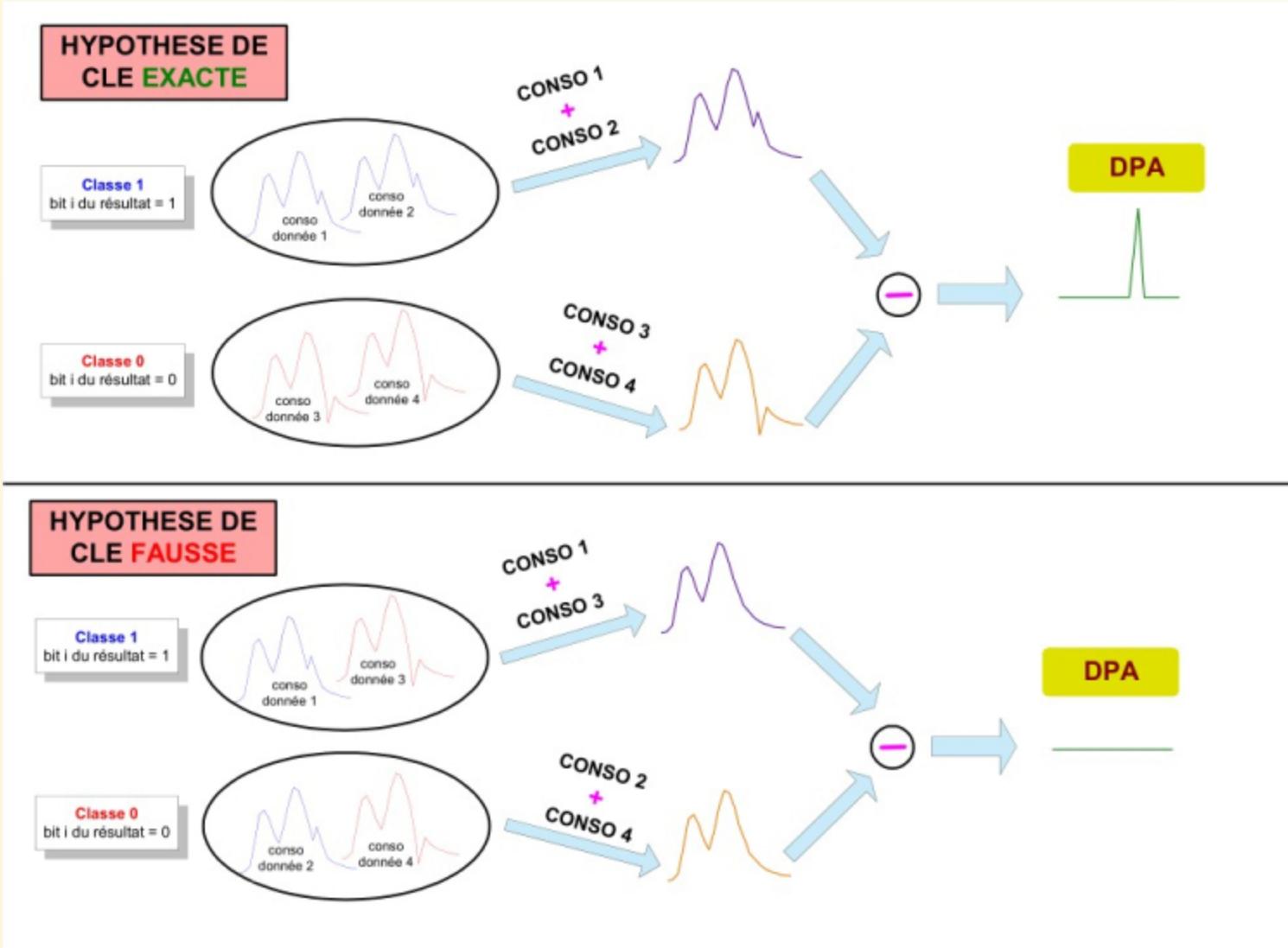
“ La redondance

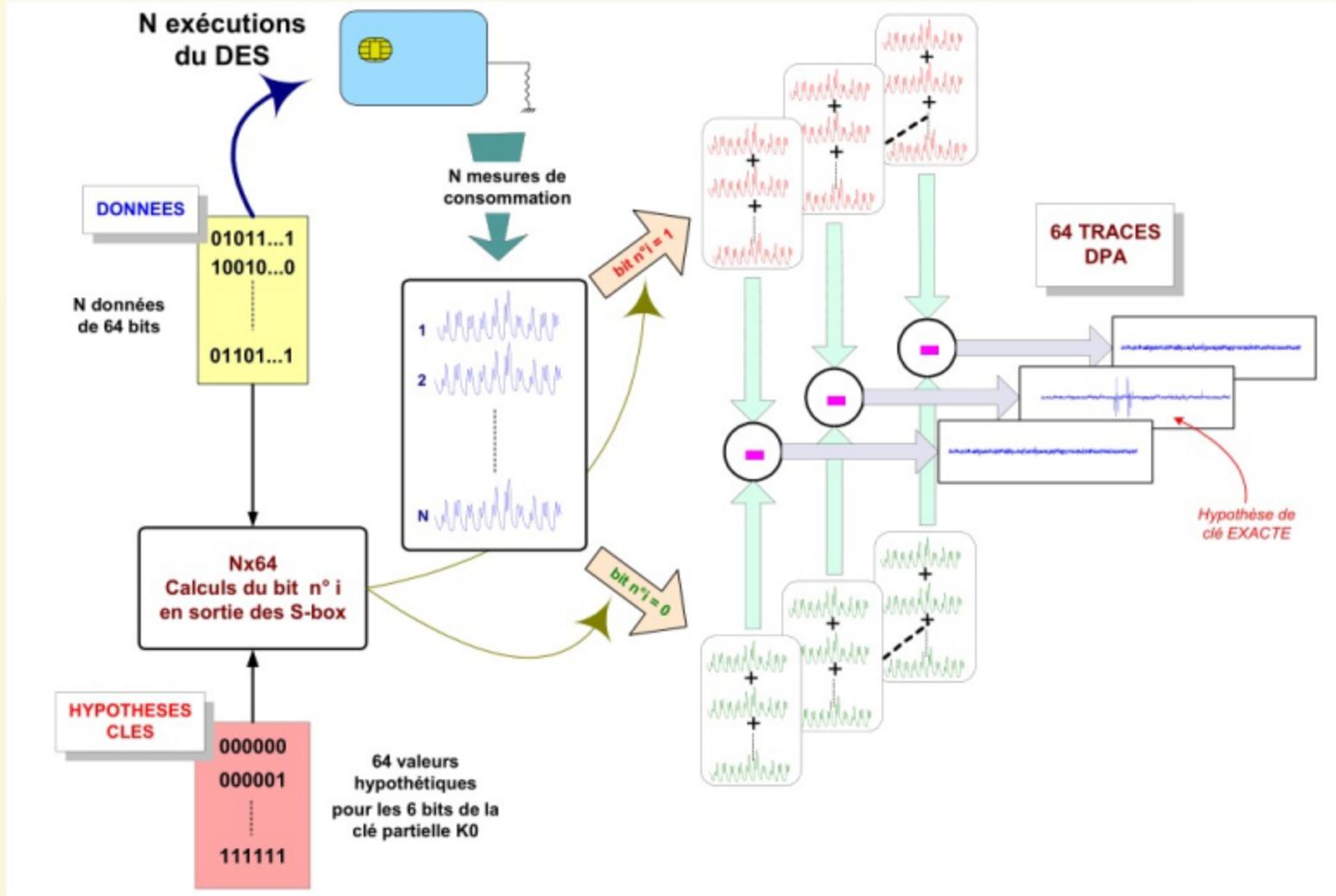
”



Side channel (ou l'art d'écouter aux portes)







Une solution

“ le jitter ... + crypto

”



Exploitation de bugs

Impact?

- Dump / injection de commande

But recherché :

- Get root → élévation de privilège
- Security bypass → authentification
- Extraction d'information → confidentialité
- Homebrew → intégrité d'exécution
- Shellcode → prise de contrôle

Les bugs ne sont pas nos amis



Stack buffer overflow

```
void foobar(int x) {  
    int local_array[7];  
    local_array[x] = 0;  
}
```

```
int main() {  
    foobar(15);  
    return 0;  
}
```

```
void foobar(int x) {  
    int local_array[7];  
    // verify the parameter is in range  
    if (x >= 0 && x < 7) {  
        local_array[x] = 0; }  
}
```

Use after free

```
int foo() {  
    int *x = (int *)malloc(4);  
    *x = 10;  
    free(x);  
    return *x;  
}
```



Signedness error example

```
char buf[128];
int len = read_len();
int buf_size = sizeof(buf);

if (len < buf_size) {
    // error and exit
}
memcpy(buf, input_len);
```

Use uninitialized

```
struct s {
    int a;
    int b;
};

int main() {
    struct s x;
    x.b = 0;
    return x.a;
}
```

Null pointer dereference

```
int global;

int *xmalloc() {
    if (global) return &global;
    return 0; // xmalloc() may return NULL
}

void npd_func_must(int flag, char *arg) {
    int *p = xmalloc(); // xmalloc() may return NULL
    *p = 1; // pointer is dereferenced without validation
}
```

Format string

```
int main()
{
    printf(gettext("This should be OK"));
    printf(some_unknown_function("This is suspicious"));
}
```

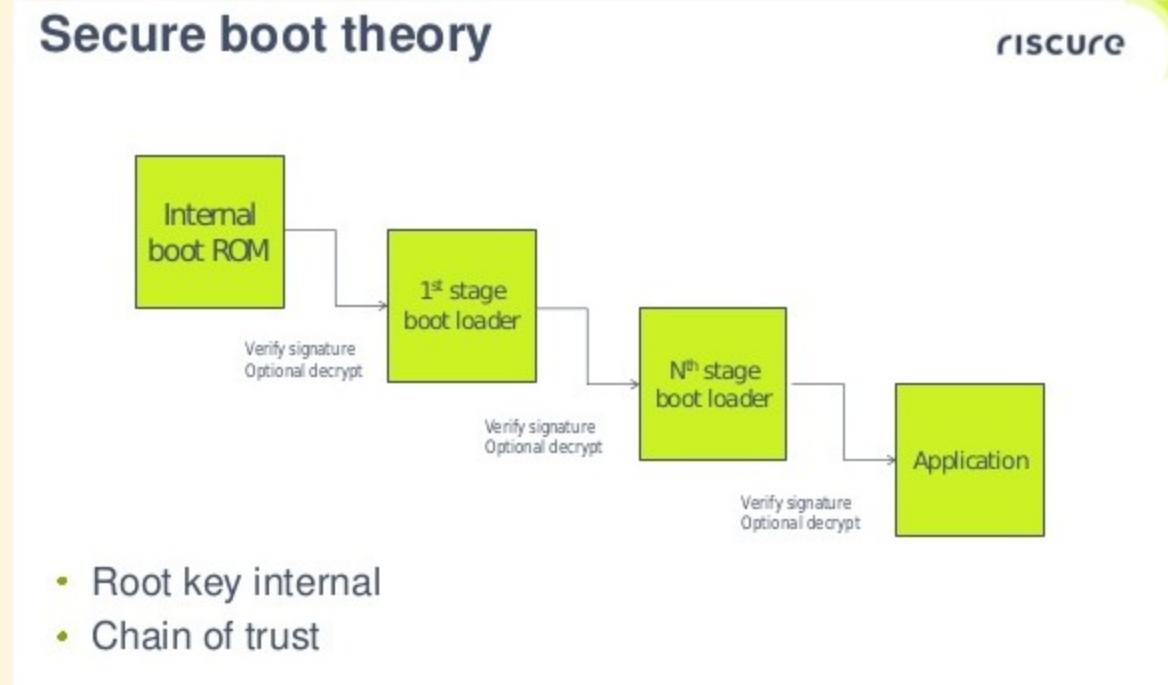
Défenses

- Hardware
 - Capteurs, fuse bits
- Architecture
 - Crypto, checksum, isolation
- Code: options de compilation
 - Canaris / Aslr / Safeseh / Dep / Got

Défenses (on n'en a jamais assez)



Secure bootloader



Crypto



Appendix C: Minimum Key Sizes and Equivalent Key Strengths

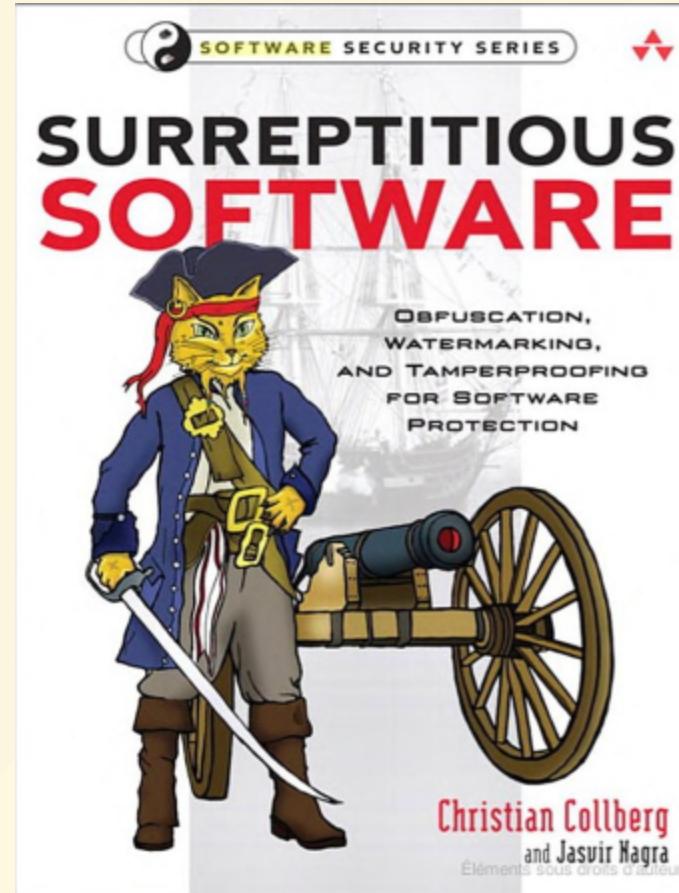
The following are the minimum key sizes and parameters for the algorithm(s) in question that must be used in connection with key transport, exchange, or establishment and for data protection:

Algorithm	DES	RSA	Elliptic Curve	DSA/D-H	AES
Minimum key size in number of bits:	112	1024	160	1024/160	128

A key-encipherment key shall be at least of equal or greater strength than any key that it is protecting. This applies to any key-encipherment key used for the protection of secret or private keys that are stored or for keys used to encrypt any secret or private keys for loading or transport. For purposes of this requirement, the following algorithms and key sizes by row are considered equivalent.

Algorithm	DES	RSA	Elliptic Curve	DSA/D-H	AES
Minimum key size in number of bits:	112	1024	160	1024/160	-
Minimum key size in number of bits:	168	2048	224	2048/224	-
Minimum key size in number of bits:	-	3072	256	3072/256	128
Minimum key size in number of bits:	-	7680	384	7680/384	192
Minimum key size in number of bits:	-	15360	512	15360/512	256

Code obfuscation / anti debugging



[Source](#)

Résumons

“ Comprendre les lois qui régissent le système

Comprendre ce que peut faire l'attaquant

Introduire les sécurités et comprendre leur limites

”

Se méfier des bugs



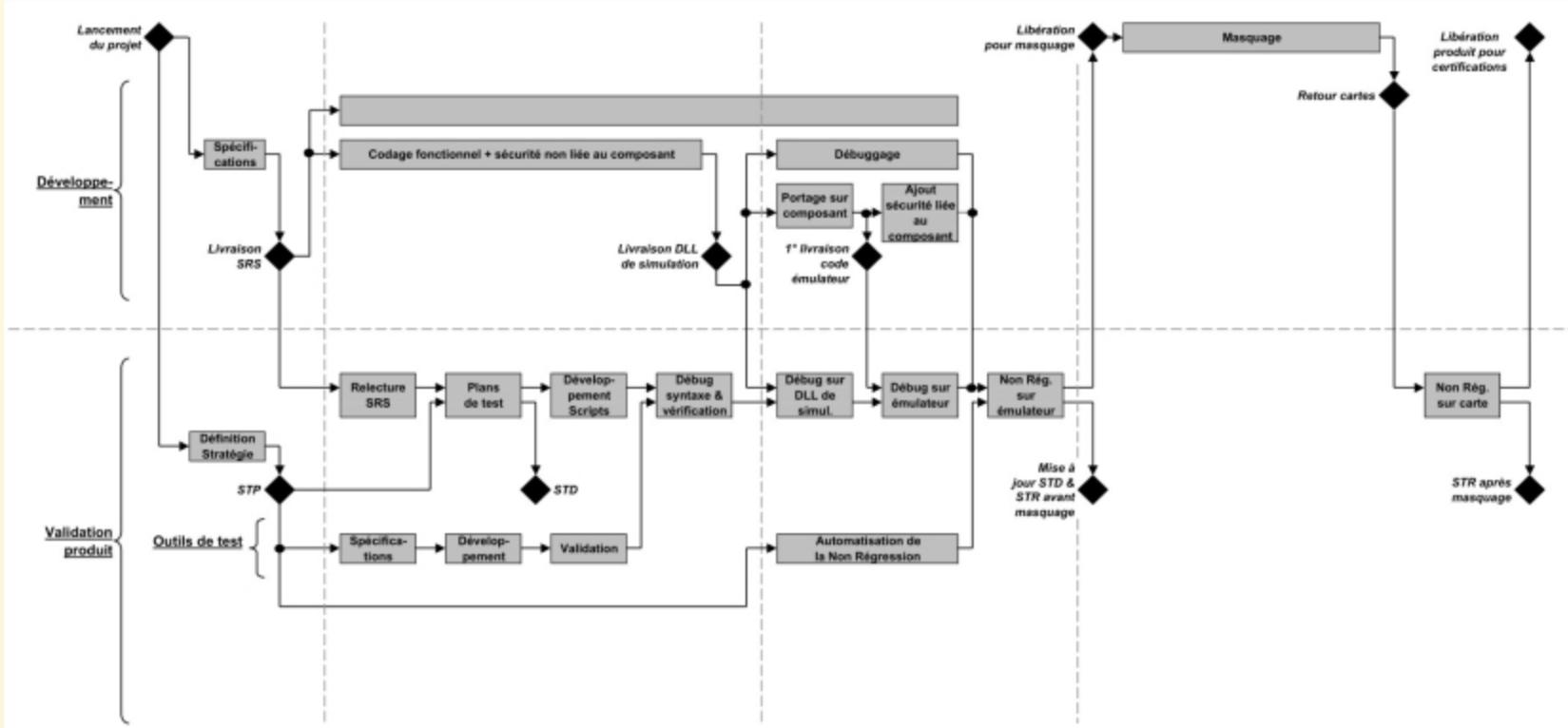
Non vraiment ... se méfier des bugs

```
int getRandomNumber()
{
    return 4; // chosen by fair dice roll.
              // guaranteed to be random.
}
```

3 - COMMENT ?-- Les soluces

Sécurité checklist

- *Produit + Surface d'attaque*
- *Qui nous attaque + Pourquoi*
- *Risque de sécurité*
- *Menace / Attaque*
- Comprendre les règles du système
- Attaques?
- **Process / compliance**



Boite à outils

- * Dev: Eclipse Git Vim ctags cscope
- * Doc: Doxygen Graphviz AStyle
- * Bugtracking: Mantis
- * Search: Ack, Opengrok
- * Metrics: Source monitor
- * Analysis: Lint, Splint, Klocwork, Cppcheck
- * Simulation: Qemu
- * Debug: GDB
- * Test: TDD, gcov, gprof, Parasoft
- * CTI: Jenkins
- * Fuzzing: Sulley, Peach, Radamsa
- * License: Blackduck
- * Reverse: Binwalk, volatility, IDA, Radare

Scope et limites de chacun

- Comprendre le besoin et la rentabilité
- Service > Coût
- Reporting



Défendre un projet

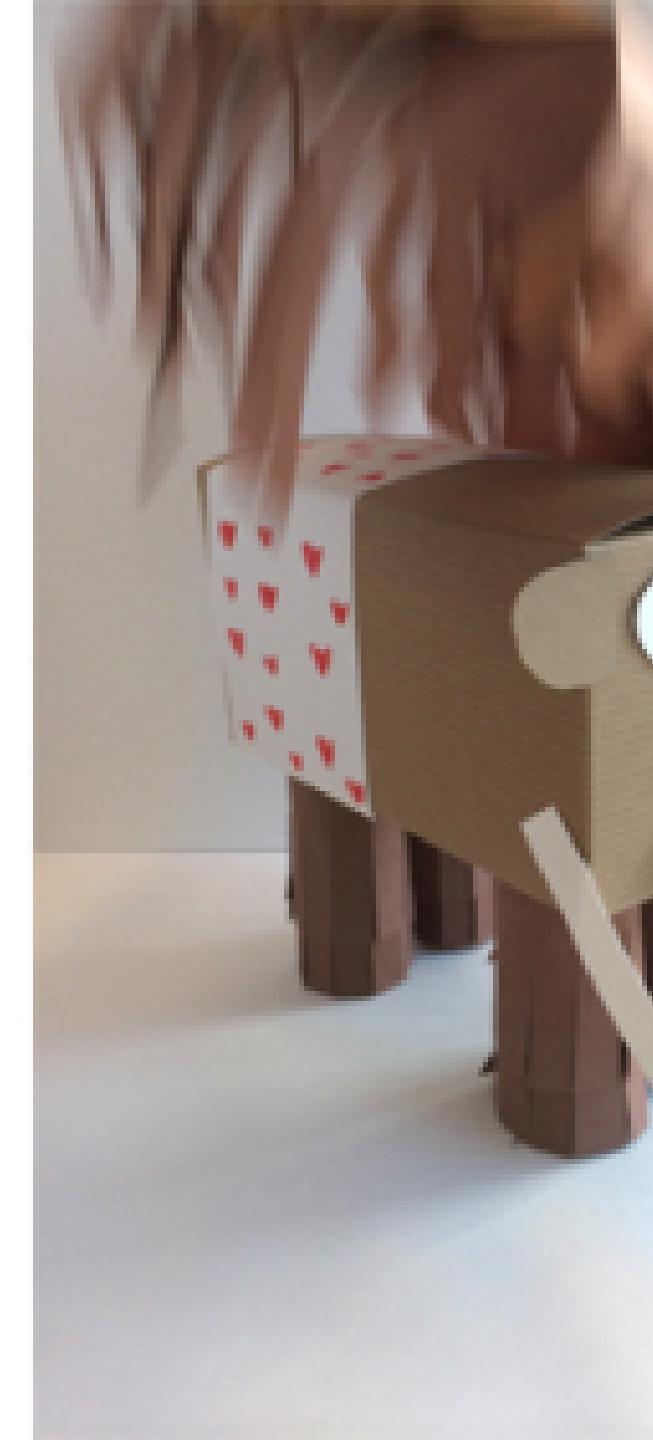
- From scratch ➔ Test, test, TEST, TEST!!
- Legacy
 - Git log / diff sont vos amis
 - Métriques ... et TEST!!



Dégrossir le projet

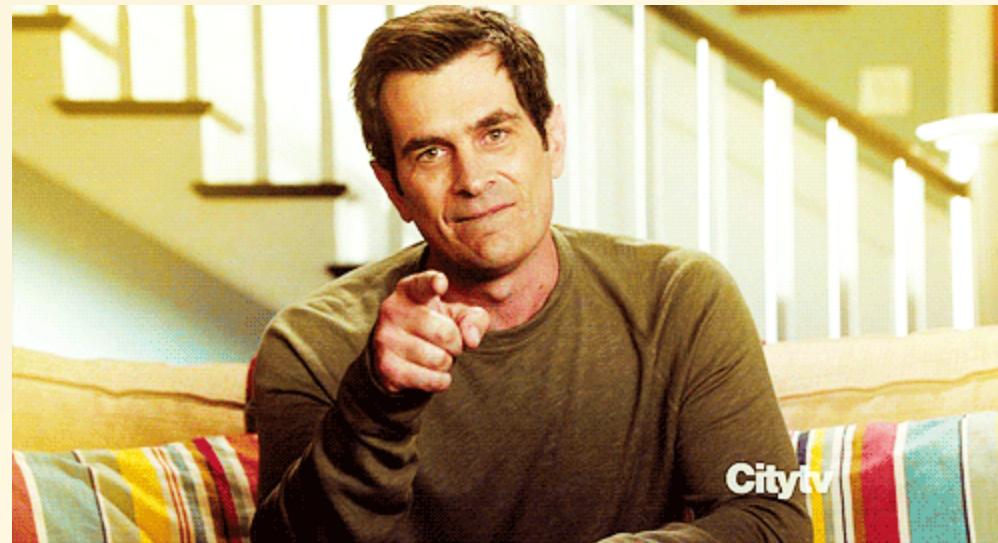
“ Where are ze bugz? ”

- Métriques
- Embarqué rechercher
 - Volatile, static
 - memcpy, printf
(équivalent)
- Profiling



Des métriques saines

- Functions ~ 200 lines
- Commentaires ~ 50 %
- Complexité cyclomatique localisée
- Taux de couverture ~ 85 %



Un environnement maîtré

- Builds reproductibles
- Tests / analyse / reporting automatique



Test embarqué

- Jtag et printf
- Emulateur / simulateur
- Par la machine ou externe
- White box / grey box / black box
- Assets / prop secu / enforcing

“ Si c'est pas testé c'est que ça marche pas ”

Compiler pour la sécurité

Gcc

- Wall -Wextra
- Wconversion -Wsign-conversion
- Wformat security
- Werror



Compiler pour la sécurité

```
-arch x86_64  
-fstack-protector-all -Wstack-protector  
--param ssp-buffer-size=4  
-pie -fPIE  
-ftrapv  
-D_FORTIFY_SOURCE=2 -O2  
-Wl,-z,relro,-z,now
```

[Source](#)

Technique (1/3)

- Ne pas être victime du **reverse engineering**
- Evaluer les **capacités de l'attaquant**
 - Extraction de binaire (elec)
 - Analyse de binaire / reverse engineering
 - Side channel / attaques par fautes
- Activer / implémenter les **contre-mesures** nécessaires au niveau **hardware / software**
- Se baser sur une **architecture saine**
 - Bootloader / crypto / communication / crypto

Technique (2/3)

- Ne pas être victime des **BUGS**
- Test, test, **TEST!**
- Utiliser les bons **outils** / comprendre leur limites
- // Couverture de code en embarqué
- Activer les **mitigations** pour limiter les dégats

Technique (3/3)

- **Prendre soin** de son projet
- Bonnes habitudes de programmation
- Eviter les pièges
- **Savoir où chercher les erreurs**
 - Les interfaces
 - Les méthodes complexes
 - Le code peu solicité

Résumons

“ Il n'existe pas de "super outil"

Mieux vaut prévenir que guérir

Interfaçage / test / analyse

Sécurité dans sa globalité

”

Pause : Git visualisation



Source: [Gource](#)

TD

Env TD

- Linux / Kali
- Tools
 - Binwalk, gcc, gdb
 - Qemu
- Doc
 - Cheatsheets

Write Ups

- Exemple: [simple writeup](#)
- **README pour explication résolution TD**
- Si complexe / perdu se ramener à un cas simple et faire des essais

TD time!



TD1 Reverse engineering et protection

- Extraction données avec binwalk
[devttyS0 devttyS1](#)
 - cible: [linux qemu 1](#)
 - out: /freeelectron/freeelectron.out
- ➔ Reverse engineering / cracking [emily](#)
 - /pc/program & program.crack & compare.sh
 - /pc/program.checksum & check.sh
- Protect avec un chiffrement / checksum
- Protect avec une signature (openssl)

TD2 Compilation attaques et défenses

- Portage embarqué => qemu
 - hello world qemu => .gdbinit
 - linux qemu 1
 - linux qemu 2
 - ➔ forme de code vs attack
 - if switch for while & x.asm & x.glitch

TD2 Compilation attaques et défenses (suite)

- Faire un bench des options de compilation gcc de defense
 - def_x_on.dump & def_x_off.dump
 - tester / montrer les limites => .gdbinit
 - ➔ changer le degré d'optimisation
 - la même chose avec qemu => .gdbinit
 - aller plus loin corelan

TD3 Sécurité d'un projet

- Auditer project
 - lister toutes les fonctions d'un programme
 - trouver la méthode la plus complexe
 -  tester la boîte à outil
 - écrire les tests fonctionnels [TDD](#)
 - aux limites / [fuzzing in memory](#)
 - aller plus loin tainting avec [splint](#), fuzzing avec [sulley](#)

TD4 Sécurité d'un projet (à emporter)

- Specs fonctionnelles: input box [coding horror](#)
 -  PIN exemple
 - implementer protocole comm tlv.c
 - chiffrement [xor](#)
 - signature avec [openssl](#)
 - create bootloader.sh with signature

TD5 Next-gen tools

- Using a constraint solver to solve problems
- Application to binary analysis
- Combine with fuzzing
- Combine with reverse engineering

Cheatsheets

- [gdb](#)
- [binwalk](#)
- [openssl](#)
- [qemu](#)