Compétitions Informatique 101

Par Ian Bouchard, Félix Larose-Gervais et Simon Thiboutôt

@corb3nik @filedesless @lilc4t

Aujourd’hui, on fait un CTF

Qu’est-ce qu’un CTF

- Une série de casse-têtes (défis) à résoudre

- Chaque défi présente un concept informatique
- Le but du défi c’est de trouver un “flag”
- Un flag peut être échangé pour des points

- But : Avoir le plus de points possible

Exemple de défi de CTF

```php
<?php
  $random_number = rand() % 10;
  if ($user_input == $random_number) {
      echo $flag;
  }

?>
```

Différents domaines de CTF

- Rétro-ingénierie (assembleur)
- Rétro-ingénierie (mobile)
- Sécurité web
- Forensique
- Stéganographie
- Défi de programmation
- Crypto-monnaie
- Exploitation binaire
- Cryptographie
- “Jail escape”

Présentation d'aujourd'hui

1. Sécurité Web
2. Défi web
3. Programmation
4. Défi de programmation
5. Rétro-ingénierie / Assembleur
6. Défi assembleur
7. Mobile + démo

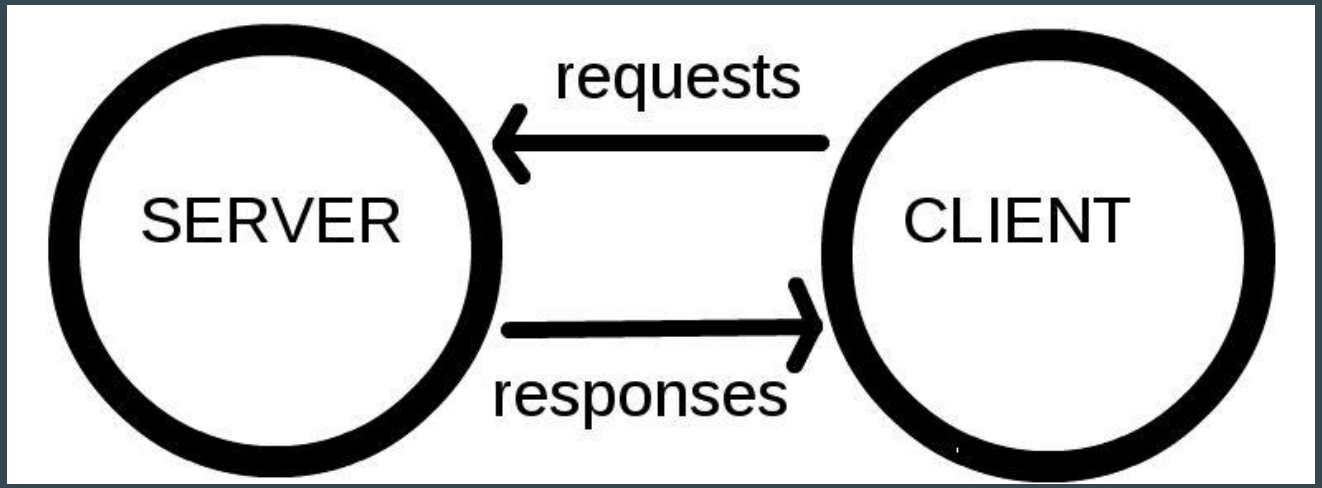
Web 101!

La Base

Le web est basé sur le protocole HTTP (HyperText Transfer Protocol).

Un client fait des requêtes à un serveur qui lui retourne des réponses.

HTTP utilise le protocole TCP (Transmission Control Protocol).



Méthodes et réponses HTTP

Liste de méthodes HTTP les plus utilisés:

- GET (demander une ressource)
- POST (transmettre des données)
- PUT (remplacer ou d'ajouter une ressource)
  PATCH (modification partielle d'une ressource)
- DELETE (supprimer une ressource)

Group de code de statut de réponse HTTP:

- Informatif : 100 à 199
- Succès : 200 à 299
- Redirection : 300 à 399
- Erreur client : 400 à 499
- Erreur de serveur : 500 à 599

Exemple (étape 1)

Vous entrez `http://example.com/` dans votre navigateur web

Le navigateur demande à un serveur DNS (Domain Name System) l'adresse IP de `example.com`.

Celui-ci répondra par exemple `93.184.216.34`

Ensuite le navigateur enverra une requête HTTP au serveur `93.184.216.34` (via TCP)

```
GET / HTTP/1.1
Host: example.com
```

Exemple (étape 2)

Le serveur va recevoir la requête et envoyer la réponse.

```
HTTP/1.1 200 OK
Content-Length: 1611
content-type: text/html; charset=UTF-8
Server: nginx
Date: Wed, 19 Sep 2018 22:35:39 GMT
Connection: close

<!DOCTYPE html>
<html>
   <body>
     <h1>Example</h1>
   </body>
</html>
```

Exemple (étape 3)

La réponse sera ensuite interprétée par votre navigateur:

- le HTML est parsé et affiché à l'écran
- les ressources (images/videos/...) vont être récupérées
- le JavaScript sera exécuté

Inspecter le code source

Lorsqu'on inspecte le code source de la page, on inspecte ce que le client a reçu comme réponse.

On voit pas le code source du serveur.

```
<!DOCTYPE html>
<html>
   <body>
     <h1>Example</h1>
   </body>
</html>
```

Extensions

On peut vérifier l'extension du fichier demandé dans l'URL.

- `http://example/index.html` Fichier HTML
- `http://example/index.php` Fichier PHP
- `http://example/index.aspx` Fichier ASP.NET
- `http://example/index.cgi` Fichier CGI (Common Gateway Interface)
- …

Headers

Les headers de la requête peuvent aussi contenir des informations importantes.

```
GET / HTTP/1.1
Host: example.com
```

```
HTTP/1.1 200 OK
Content-Length: 1611
content-type: text/html; charset=UTF-8
Server: nginx
Date: Wed, 19 Sep 2018 22:35:39 GMT
Connection: close

<!DOCTYPE html>
<html>
   <body>
     <h1>Example</h1>
   </body>
</html>
```

Custom Headers

Il est possible d'ajouter des headers custom qui ne sont pas définie dans le standard.

Les noms des headers customs sont préfixés par `X-`, exemple `X-Custom-Header`.

Liste des headers standard

Hacking

Les étapes de bases:

1. Comprendre le but de l'application
2. Récupérer l'information sur l'application
3. Trouver le but du challenge
4. Trouver le flag

Prendre des notes!

1. Comprendre le but de l’application

- Elle sert à faire quoi?
- Comment l'utiliser?

On fait le tour des pages disponibles.

On soumet les formulaires.

2. Récupérer l'information sur l'application

On cherche à comprendre comment elle fonctionne au niveau technique.

On essaye de s'imaginer comment pourrait être le code source du serveur.

- HTTP headers
  - Custom headers
  - Cookies
- Code HTML
  - `<!-- commentaires -->`
  - `<form>`
    - Attribute `action`
    - Attribute `method`
    - `<input type="hidden">`
  - …
- Code CSS
- Code JavaScript
  - Requêtes AJAX
  - Console: log, warning, error, ...
- …

3. Trouver le but du challenge

Le but du challenge n'est peut-être pas explicitement donné.

Qu'est-ce qu'on doit exploiter?

Rien n'est laissé au hasard.

Indice:

- le nom du challenge
- description du challenge

OWASP Top 10 (risques liés à la sécurité des applications)

4. Trouver le flag

Outils et ressources

Afin de réussir à faire les challenges du CFI CTF vous devez être capable d'envoyer une requête HTTP créée de toutes pièces.

L'outils Postman est parfait pour cela!

Bonus:

- Workflow d'exploitation Web par Ian Bouchard!

Live demo!

Défis

Niveau 1

Niveau 2

Programmation + sockets

Sockets?

- Interface pour interagir avec un canal de communication
- Un peu comme un fichier (**open/close**, **read/write**)
- Contrairement au fichier; viennent en paires (modèle **client/serveur**)
- Plusieurs types:
  - **Unix**
  - **UDP**
  - **TCP**

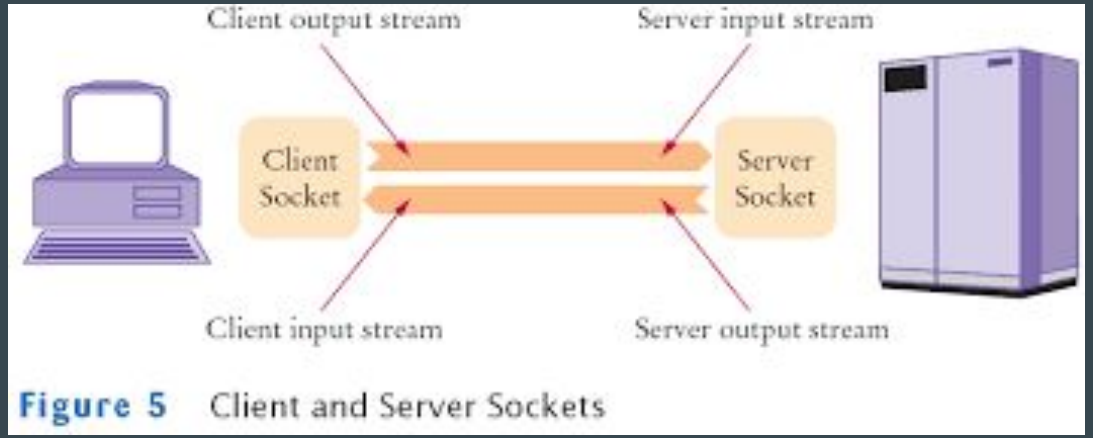
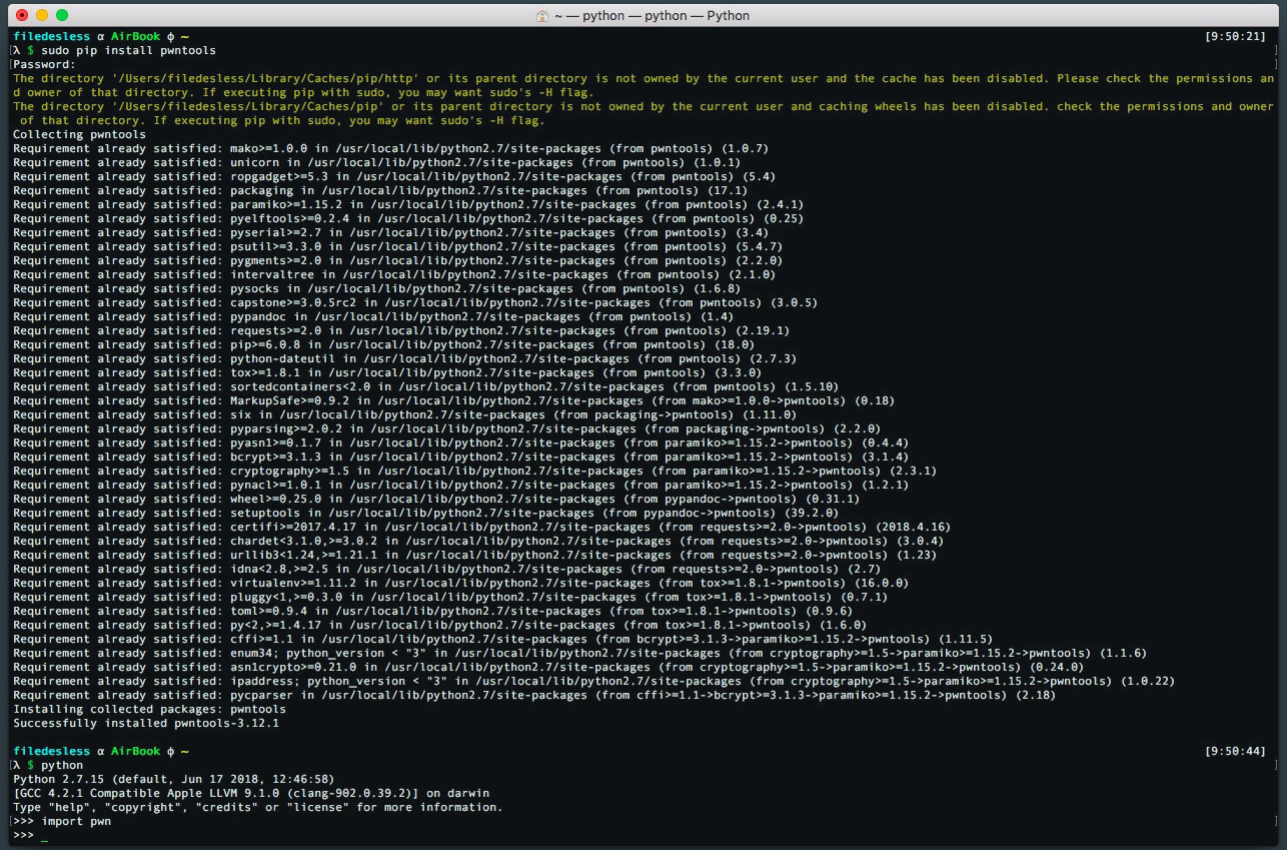


Figure 5 Client and Server Sockets

pwntools?

- Nice lib pour CTFs
- Python2 seulement
- Pip install



Exemples

==> quotd.py <==

```
#!/usr/bin/env python2

from pwn import remote

with remote("5.9.23.24", 17) as r:
    print(r.recvuntil("\n\n")
```

==> http.py <==

```
#!/usr/bin/env python2

from pwn import remote

with remote("ifconfig.co", 80) as r:
    r.sendline("GET / HTTP/1.1")
    r.sendline("HOST: ifconfig.co")
    r.sendline()

    print(r.recvuntil("\n\n"))
```

==> gopher.py <==

```
#!/usr/bin/env python2

from pwn import remote

with remote("hacking.allowed.org", 70) as r:
    r.sendline("/tools/")
    print(r.recvuntil("\n."))
```

==> irc.py <==

```
#!/usr/bin/env python2

from pwn import remote

with remote("irc.freenode.net", 6666) as r:
    r.sendline("NICK filedesless")
    r.sendline("USER filedesless * 8 :filedesless")

    print(r.recvuntil(":filedesless"))
    print("Showing whois")
```

Rétro-ingénierie + ASM (x86_64)

Exemple de défi

- Défi typique : on te donne une application compilée, tu dois comprendre/retrouver le code original.

Exemple de défi

- Défi typique : on te donne une application compilée, tu dois comprendre/retrouver le code original.

```
// ASM
mov rdi, 23
mov rsi, 44
call add
```

Exemple de défi

- Défi typique : on te donne une application compilée, tu dois comprendre/retrouver le code original.

```
// ASM
mov rdi, 23
mov rsi, 44
call add
```

```
// C
add(23,  44)
```

ASM en 5 minutes

- La majorité des instructions en ASM suit la structure suivante : instr dest, src
- Chaque instruction opère sur des registres et des constantes
- Il y a beaucoup de registres :
  - rax
  - rbx
  - rcx,
  - rdx
  - rsp, rbp,
  - rsi, rdi, r8, r9, r10, r11, r12, r13, r14, r15
  - ...

ASM en 5 minutes

- La majorité des instructions en ASM suit la structure suivante : instr dest, src

- Le comportement d'un instruction est généralement explicite :

  - mov rax, 123 # rax = 123
  - add rcx, rbx # rcx = rcx + rbx
  - jmp sub_1 # goto sub_1
  - sub rdi, r8 # rdi = rdi - r8
  - cmp rdx, 29 # comparaison entre rdx et 29
  - call rand # rand()

ASM en 5 minutes - Appeler une fonction

- On appelle une fonction avec l'instruction call
- Les arguments d'une fonction sont définis dans des registres particuliers :

  - rdi # arg 1
  - rsi # arg 2
  - rdx # arg 3
  - r8 # arg 4
  - r9 # arg 5

- La valeur de retour d'une fonction est stockée dans rax

Exemple de défi

- Défi typique : on te donne une application compilée, tu dois comprendre/retrouver le code original.

```
// ASM
mov rdi, 23
mov rsi, 44
call add
```

```
// C
add(23,  44)
```

Exercice

```
// ASM
mov rbx, 100
mov rcx, rbx
add rbx, 10
mov rsi, rbx
mov rdi, rcx
call add
```

Reminder :

- RDI => Argument #1
- RSI => Argument #2

Exercice

```
// ASM
mov rbx, 100
mov rcx, rbx
add rbx, 10
mov rsi, rbx
mov rdi, rcx
call add
```

```
// C
add(100,  100+10)
```

Reminder :

- RDI => Argument #1
- RSI => Argument #2

Outils & Ressources!

- Apprendre l'assembleur en écrivant du C : https://godbolt.org/
- Désassembleur (Linux & OSX) : https://www.hopperapp.com/
- Désassembleur (Windows) https://www.hex-rays.com/products/ida/support/download_freeware.shtml

- Liste d'instructions x86_64 : http://linasm.sourceforge.net/docs/instructions/cpu.php#data

Hopper Démo

Rétro-ingénierie + mobile (Android)

La base

Un fichier d'application Android est un fichier de type APK.

C'est en fait un fichier ZIP contenant le code compilé ainsi que les ressources de l'application.

```
$ file app.apk
App.apk: Zip archive data
```

Décompresser un APK

```
$ unzip app.apk -q -d unzip-app

$ ls -1 unzip-app/
AndroidManifest.xml
META-INF
classes.dex
res
resources.arsc
```

Décompiler un APK

On va utiliser un outil appelé JADX.

Installation

- Download
- MacOS avec brew : `$ brew install jadx`

Live demo!

Prochaines étapes

Prochaines étapes

- Entraînement personnel :
  - root-me.org (sécurité)
  - ringzer0team.com (sécurité)
  - codewars.com (programmation)
  - ...

- Prochains évènements :
  - Qualification des CSGames
  - Coveo Blitz
  - Autres suggestions?