Table des matières

Avant-Propos

Ce document est le compte rendu du TP Analyse de trame réalisé le 4 Décembre 2015. Pour réaliser ce TP, nous avons utilisé un formulaire mis en ligne par Mr.Lebatteur sur sa machine, pour analyser les trames que nos machines échangent lors de l'envoie de données. Pour ce faire, nous utilisons un logiciel gratuit qui se nomme Wireshark, et qui permet de voir les trames qui transitent sur notre carte réseau.

E6 :

Elaboration de documents relatifs à la production et à la fourniture de services
A1.1.1 , Analyse du cahier des charges d'un service à produire
A1.2.4 , Détermination des tests nécessaires à la validation d'un service
A2.2.1 , Suivi et résolution d'incidents
A4.1.9 , Rédaction d'une documentation technique

Capture des 3 trames de connexion TCP

Une connexion TCP/IP commence à chaque fois par une négociation ternaire. Le PC client (ici mon poste, avec l'adresse 192.168.1.59) envoie une requête au serveur (ici le poste de Mr.Lebatteur qui fait office de serveur web, avec l'adresse 192.168.1.75) et lui demande s'il peut se connecter. Ensuite, le serveur répond avec un accord de connexion, puis le client répond que la connexion est bien établie.

Première trame de connexion :

```
39 2.047917000 192.168.1.59    192.168.1.75    TCP    66 49403→80 [SYN] Seq=0 Win=8192 Len=0 MSS=1452 WS=16 S
```

Cette première trame est la première étape de la négociation ternaire. On peut voir sur la capture d'écran ci-dessous que la source est bien ma machine, et que la destination est celle du serveur.

```
Source: 192.168.1.59 (192.168.1.59)
Destination: 192.168.1.75 (192.168.1.75)
```

Pour reconnaitre la première trame de la négociation ternaire, il faut se fier au bit de contrôle, qui est TOUJOURS à 1 sur le Syn, donc sur la synchronisation des numéros de séquence. Comme on peut le voir ci-dessous, ou sur ce que Wireshark interprète, le SYN est bien à 1, c'est donc la première étape de la négociation ternaire.

```
000. .... .... = Reserved: Not set
...0 .... .... = Nonce: Not set
.... 0... .... = Congestion Window Reduced (CWR): Not set
.... .0.. .... = ECN-Echo: Not set
.... ..0. .... = Urgent: Not set
.... ...0 .... = Acknowledgment: Not set
.... .... 0... = Push: Not set
.... .... .0.. = Reset: Not set
.... .... ..1. = Syn: Set
```

Deuxième trame :

```
42 2.048714000 192.168.1.75    192.168.1.59    TCP    66 80→49403 [SYN, ACK] Seq=0 Ack=1
```

Cette deuxième trame est l'accord de connexion du serveur vers la machine. Le serveur veut donc bien que ma machine se connecte sur un de ces ports, on peut voir sur la capture d'écran que la source est bien le serveur et que la destination est bien ma machine.

```
Source: 192.168.1.75 (192.168.1.75)
Destination: 192.168.1.59 (192.168.1.59)
```

Lors de la deuxième étape, les bits Syn et Acknowledgment(Ack) sont à un, le serveur accepte donc la demande de connexion. Ces bits sont toujours à 1 lors de la deuxième étape.

```
000. .... .... = Reserved: Not set
...0 .... .... = Nonce: Not set
.... 0... .... = Congestion Window Reduced (CWR): Not set
.... .0.. .... = ECN-Echo: Not set
.... ..0. .... = Urgent: Not set
.... ...1 .... = Acknowledgment: Set
.... .... 0... = Push: Not set
.... .... .0.. = Reset: Not set
.... .... ..1. = Syn: Set
```

Troisième et dernière étape :

```
44 2.048836000 192.168.1.59 192.168.1.75 TCP 54 49403→80 [ACK] Seq=1 Ack
```

Cette troisième trame est envoyée par le client vers le serveur, et permet au client de valider que le serveur as bien validé la demande de connexion. On peut voir que la source est bien ma machine et que la destination est bien le serveur.

```
Source: 192.168.1.59 (192.168.1.59)
Destination: 192.168.1.75 (192.168.1.75)
```

Lors de cette dernière étape de la négociation ternaire, Acknowledgment est à 1, ce qui est toujours le cas lors de la dernière étape.

```
000. .... .... = Reserved: Not set
...0 .... .... = Nonce: Not set
.... 0... .... = Congestion Window Reduced (CWR): Not set
.... .0.. .... = ECN-Echo: Not set
.... ..0. .... = Urgent: Not set
.... ...1 .... = Acknowledgment: Set
.... .... 0... = Push: Not set
.... .... .0.. = Reset: Not set
.... .... ..0. = Syn: Not set
.... .... ...0 = Fin: Not set
```

Cette négociation ternaire est faite à chaque fois que le client as besoin d'une information sur le serveur, et cette opération est répétée à chaque fois. C'est une étape indispensable dans la connexion TCP/IP.

Capture des trames qui affichent le formulaire HTML

(Il est important de regarder les numéros de ports, ils sont attribués automatiquement (dynamiquement) et peuvent changer, mais pas dans la même discutions)

Maintenant que le client et le serveur sont connectés, l'échange de donnés peut enfin commencer. Le client (l'adresse IP qui se trouve tout à gauche) émet une requête http au serveur (l'adresse juste à côté, a sa droite, toujours la même elle aussi) pour récupérer une page HTML, avec l'adresse suivant : /ppe/login.php

```
47 2.322527000 192.168.1.59  192.168.1.75  HTTP  426 GET /ppe/login.php HTTP/1.1
48 2.322817000 192.168.1.75  192.168.1.59  TCP    60 80→49403 [ACK] Seq=1 Ack=373 Win=66784 Len=0
49 2.325367000 192.168.1.75  192.168.1.59  HTTP  662 HTTP/1.1 200 OK  (text/html)
50 2.325403000 192.168.1.59  192.168.1.75  TCP    54 49403→80 [ACK] Seq=373 Ack=609 Win=66176 Len=0
```

On peut voir ici les trames permettant d'afficher le code source de la page. Dans la première trame, le client demande au serveur de lui donner la page /ppe/login.php, ce que l'on a rentré dans le navigateur pour accéder à la page

192.168.1.75/ppe/login.php

Le serveur répond en disant qu'il a bien reçu la demande, avec un Acknowledgment puis envoie le code source de la page. Pour finir, le client répond en disant qu'il a bien reçu le code source.

La troisième trame contient le code source de la page, que l'on peut rechercher en cherchant dans les onglets interprétés par Wireshark. Il faut chercher dans le bon onglet, le « Line-baser test data : text/html »

```
⊞ Frame 49: 662 bytes on wire (5296 bits), 662 bytes captured (5296 bits) on interface 0
⊞ Ethernet II, Src: Giga-Byt_e6:bb:f0 (74:d4:35:e6:bb:f0), Dst: Giga-Byt_e6:7f:70 (74:d4:35:e6:7f:70)
⊞ Internet Protocol Version 4, Src: 192.168.1.75 (192.168.1.75), Dst: 192.168.1.59 (192.168.1.59)
⊞ Transmission Control Protocol, Src Port: 80 (80), Dst Port: 49403 (49403), Seq: 1, Ack: 373, Len: 608
⊞ Hypertext Transfer Protocol
⊟ Line-based text data: text/html
    <html>\r\n
    <head>\r\n
    <title>Page d'identification</title>\r\n
    </head>\r\n
    <body>\r\n
    <form name="form" method="POST" action="testlogin.php">\r\n
    Login : <input type="text" name="login" /><br />\r\n
    Mot de passe : <input type="password" name="mdp" />\r\n
    <br /><br />\r\n
    <input type="submit" name="valider" value="Valider" />\r\n
    <input type="reset" name="Annuler" value="Annuler" />\r\n
    </form>\r\n
    </body>\r\n
    </html>\r\n
```

Le code source de la page est bien du html. Ce qui est marqué ici est ce qui est interprété par Wireshark, en effet les données normalement contenues dans la trame sont en hexadécimal, et voici a quoi cela ressemble :

```
0000  74 d4 35 e6 7f 70 74 d4  35 e6 bb f0 08 00 45 00   t.5..pt. 5.....E.
0010  02 88 67 82 40 00 80 06  0d 17 c0 a8 01 4b c0 a8   ..g.@... .....K..
0020  01 3b 00 50 c1 9c 3c a1  a4 a1 6b ad 95 73 50 18   .;.P..<. ..k..sP.
0030  10 4e b4 ed 00 00 48 54  54 50 2f 31 2e 31 20 32   .N....HT TP/1.1 2
0040  30 30 20 4f 4b 0d 0a 44  61 74 65 3a 20 46 72 69   00 OK..D ate: Fri
0050  2c 20 30 34 20 44 65 63  20 32 30 31 35 20 31 30   , 04 Dec  2015 10
0060  3a 30 39 3a 32 39 20 47  4d 54 0d 0a 53 65 72 76   :09:29 G MT..Serv
0070  65 72 3a 20 41 70 61 63  68 65 2f 32 2e 34 2e 39   er: Apac he/2.4.9
0080  20 28 57 69 6e 36 34 29  20 50 48 50 2f 35 2e 35    (Win64)  PHP/5.5
0090  2e 31 32 0d 0a 58 2d 50  6f 77 65 72 65 64 2d 42   .12..X-P owered-B
00a0  79 3a 20 50 48 50 2f 35  2e 35 2e 31 32 0d 0a 43   y: PHP/5 .5.12..C
00b0  6f 6e 74 65 6e 74 2d 4c  65 6e 67 74 68 3a 20 33   ontent-L ength: 3
00c0  38 33 0d 0a 4b 65 65 70  2d 41 6c 69 76 65 3a 20   83..Keep -Alive:
00d0  74 69 6d 65 6f 75 74 3d  35 2c 20 6d 61 78 3d 31   timeout= 5, max=1
00e0  30 30 0d 0a 43 6f 6e 6e  65 63 74 69 6f 6e 3a 20   00..Conn ection:
00f0  4b 65 65 70 2d 41 6c 69  76 65 0d 0a 43 6f 6e 74   Keep-Ali ve..Cont
0100  65 6e 74 2d 54 79 70 65  3a 20 74 65 78 74 2f 68   ent-Type : text/h
0110  74 6d 6c 0d 0a 0d 0a 3c  68 74 6d 6c 3e 0d 0a 3c   tml....< html>..<
0120  68 65 61 64 3e 0d 0a 3c  74 69 74 6c 65 3e 50 61   head>..< title>Pa
0130  67 65 20 64 27 69 64 65  6e 74 69 66 69 63 61 74   ge d'ide ntificat
0140  69 6f 6e 3c 2f 74 69 74  6c 65 3e 0d 0a 3c 2f 68   ion</tit le>..</h
0150  65 61 64 3e 0d 0a 3c 62  6f 64 79 3e 0d 0a 3c 66   ead>..<b ody>..<f
0160  6f 72 6d 20 6e 61 6d 65  3d 22 66 6f 72 6d 22 20   orm name ="form"
0170  6d 65 74 68 6f 64 3d 22  50 4f 53 54 22 20 61 63   method=" POST" ac
0180  74 69 6f 6e 3d 22 74 65  73 74 6c 6f 67 69 6e 2e   tion="te stlogin.
0190  70 68 70 22 3e 0d 0a 4c  6f 67 69 6e 20 3a 20 3c   php">..L ogin : <
01a0  69 6e 70 75 74 20 74 79  70 65 3d 22 74 65 78 74   input ty pe="text
01b0  22 20 6e 61 6d 65 3d 22  6c 6f 67 69 6e 22 20 2f   " name=" login" /
01c0  3e 3c 62 72 20 2f 3e 0d  0a 4d 6f 74 20 64 65 20   ><br />. .Mot de
01d0  70 61 73 73 65 20 3a 20  3c 69 6e 70 75 74 20 74   passe :  <input t
01e0  79 70 65 3d 22 70 61 73  73 77 6f 72 64 22 20 6e   ype="pas sword" n
01f0  61 6d 65 3d 22 6d 64 70  22 20 2f 3e 0d 0a 3c 62   ame="mdp " />..<b
0200  72 20 2f 3e 3c 62 72 20  2f 3e 0d 0a 3c 69 6e 70   r /><br  />..<inp
0210  75 74 20 74 79 70 65 3d  22 73 75 62 6d 69 74 22   ut type= "submit"
0220  20 6e 61 6d 65 3d 22 56  61 6c 69 64 65 72 22 20    name="V alider"
0230  76 61 6c 75 65 3d 22 56  61 6c 69 64 65 72 22 20   value="V alider"
0240  2f 3e 0d 0a 3c 69 6e 70  75 74 20 74 79 70 65 3d   />..<inp ut type=
0250  22 72 65 73 65 74 22 20  6e 61 6d 65 3d 22 41 6e   "reset"  name="An
0260  6e 75 6c 65 72 22 20 76  61 6c 75 65 3d 22 41 6e   nuler" v alue="An
0270  6e 75 6c 65 72 22 20 2f  3e 0d 0a 3c 2f 66 6f 72   nuler" / >..</for
0280  6d 3e 0d 0a 3c 2f 62 6f  64 79 3e 0d 0a 3c 2f 68   m>..</bo dy>..</h
0290  74 6d 6c 3e 0d 0a                                  tml>..
```

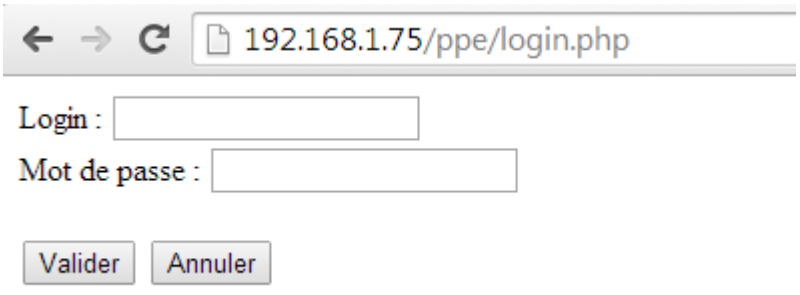
Ce qui est surligné est le numéro de ligne, donc on n'en tiens pas compte. Tout le reste au milieu est ce qui est contenu dans la trame, et à droite ce qui est traduit par Wireshark. On peut voir que c'est très long, et que le logiciel fait tout le travail de conversion à notre place, ce qui est très pratique ! Voici donc le code source de la page traduit par Wireshark :

```
Line-based text data: text/html
    <html>\r\n
    <head>\r\n
    <title>Page d'identification</title>\r\n
    </head>\r\n
    <body>\r\n
    <form name="form" method="POST" action="testlogin.php">\r\n
    Login : <input type="text" name="login" /><br />\r\n
    Mot de passe : <input type="password" name="mdp" />\r\n
    <br /><br />\r\n
    <input type="submit" name="Valider" value="Valider" />\r\n
    <input type="reset" name="Annuler" value="Annuler" />\r\n
    </form>\r\n
    </body>\r\n
    </html>\r\n
```

Et voici ce que cela donne dans le code source de la page dans notre navigateur :

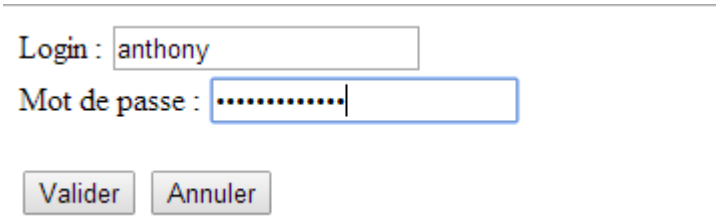
```
<html>
<head>
<title>Page d'identification</title>
</head>
<body>
<form name="form" method="POST" action="testlogin.php">
Login : <input type="text" name="login" /><br />
Mot de passe : <input type="password" name="mdp" />
<br /><br />
<input type="submit" name="Valider" value="Valider" />
<input type="reset" name="Annuler" value="Annuler" />
</form>
</body>
</html>

```

Et voilà ce qu'il s'affiche dans notre navigateur, un formulaire ou l'on peut remplir les champs et valider pour envoyer un login et un mot de passe.

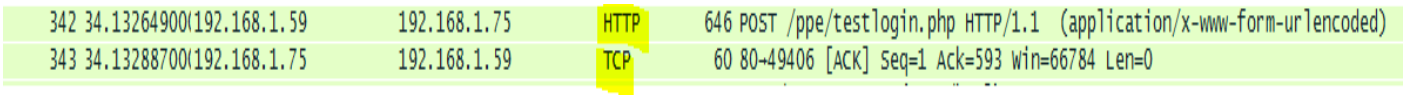


Trames qui affichent de le mot de passe et le login en clair

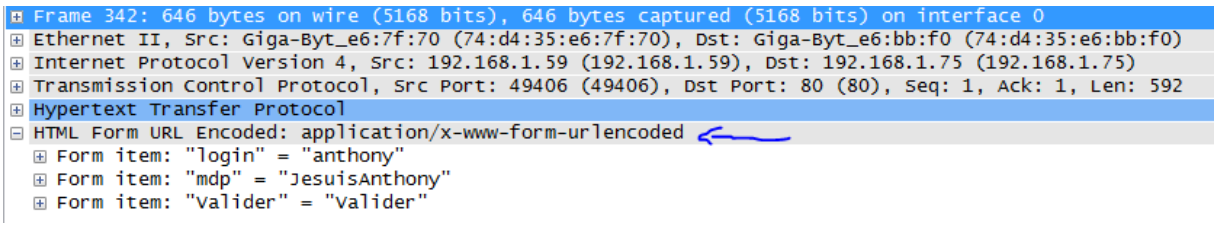
Une fois que les champs sont remplis, on clique sur valider, et les informations sont envoyées.



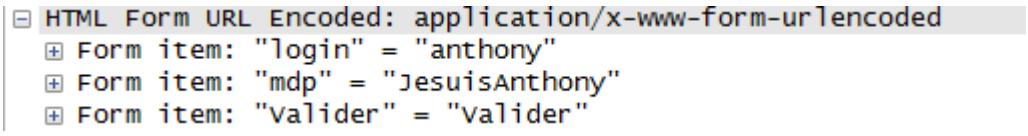
Sur Wireshark, on peut voir que la machine client envoie vers le serveur à l'appui du bouton valider, les données du login et du mot de passe sont envoyées, et ne sont pas cryptées.
On peut voir que le client envoie en protocole http les données des champs, et qu'il est très facile de les voir. Le serveur répond avec un Acknowledgment pour valider le fait qu'il ait bien reçu les données.



Dans la première trame, celle en http, on peut voir dans l'onglet HTML Form URL Encoded les données que l'on a rentrées sur la page html.



On retrouve biens les champs login et mdp, avec les informations que j'ai rentrées dedans, soit comme login « anthony » et comme mot de passe « JesuisAnthony ».



Login et mot de passe dans le script PHP

Maintenant que le données sont envoyées par le client, le serveur envoie une nouvelle page html, avec du code source, contenant le login et le mot de passe que l'on a rentré.

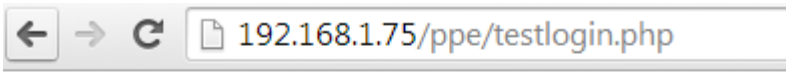
```
344 34.13613700(192.168.1.75    192.168.1.59    HTTP    408 HTTP/1.1 200 OK  (text/html)
345 34.13618600(192.168.1.59    192.168.1.75    TCP      54 49406→80 [ACK] Seq=593 Ack=355 Win=66432 Len=0
```

On peut voir que dans la première trame, il y'as le code source de la page, qui est traité par notre navigateur pour afficher la page.

```
Line-based text data: text/html
    <html>\r\n
    <head>\r\n
    <title>Test login</title>\r\n
    </head>\r\n
    <body>\r\n
    Login : anthony<br />\r\n
    Mot de passe : JesuisAnthony</body>\r\n
    </html>\r\n
```

```
1 <html>
2 <head>
3 <title>Test login</title>
4 </head>
5 <body>
6 Login : anthony<br />
7 Mot de passe : JesuisAnthony</body>
8 </html>
9
```

La dernière trame est envoyée par le client au serveur pour valider le fait qu'il ait bien reçu la page html. La page qui s'affiche est la suivante :



Login : anthony
Mot de passe : JesuisAnthony

On y retrouve donc les informations que j'ai rentrées.

Trames de déconnexion

Maintenant que l'on a reçu les données, nous n'avons plus rien à faire avec le serveur, nous pouvons donc mettre fin à la communication

Première trame de déconnexion :

```
32 4.313208000 192.168.1.59    192.168.1.75    TCP    54 49565→80 [FIN, ACK] Seq=1 Ack=1 Win=66784 Len=0
```

Cette trame est envoyée par le client au serveur, pour dire qu'il veut se déconnecter, avec le bit de contrôle FIN, il veut donc mettre fin à la communication.

```
  000. .... .... = Reserved: Not set
  ...0 .... .... = Nonce: Not set
  .... 0... .... = Congestion Window Reduced (CWR): Not set
  .... .0.. .... = ECN-Echo: Not set
  .... ..0. .... = Urgent: Not set
  .... ...1 .... = Acknowledgment: Set
  .... .... 0... = Push: Not set
  .... .... .0.. = Reset: Not set
  .... .... ..0. = Syn: Not set
⊟ .... .... ...1 = Fin: Set
```

Deuxième trame de déconnexion :

Cette deuxième trame est envoyée par le serveur et acquitte de la fermeture de la connexion.

```
34 4.313437000 192.168.1.75    192.168.1.59    TCP    60 80→49565 [ACK] Seq=1 Ack=2 Win=
```

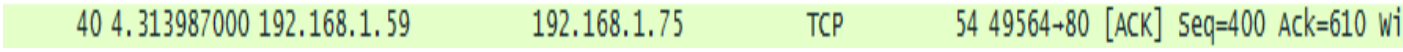
Troisième trame de déconnexion :

Cette troisième trame envoyée par le serveur au client refait une demande de déconnexion.

```
38 4.313857000 192.168.1.75    192.168.1.59    TCP    60 80→49564 [FIN, ACK] Seq=609 Ack=400 Win=66784 Len=0
```

Quatrième trame de déconnexion :

Cette quatrième trame est envoyée par le client et valide la demande de déconnexion du serveur.



Maintenant la communication est terminée. Il ne faut pas laisser une communication trop longtemps, car c'est une énorme faille de sécurité, c'est pour cela qu'au bout de quelques secondes il y'as automatiquement une demande de déconnexion, et si il doit encore une fois y avoir échange de données, alors une demande de reconnexion est faite.