

HEIG-VD

SÉCURITÉ DES RÉSEAUX SANS fil

LABORATOIRE 01

Laboratoire 802.11 sécurité MAC

Auteurs

VALZINO BENJAMIN

TISSOT OLIVIER

BAILAT JOACHIM

Professeur

RUBINSTEIN MARCOS

02-04-2023

HE
IG

Table des matières

| | |
|---|----------|
| Partie 1 - beacons, authentication | 3 |
| Deauthentication attack | 3 |
| Question 1 | 3 |
| Question 2 | 3 |
| Question 3 | 4 |
| Question 4 | 4 |
| Question 5 | 4 |
| Question 6 | 4 |
| Question 7 | 5 |
| Fake channel evil tween attack | 5 |
| Question 1 | 5 |
| SSID flood attack | 6 |
| Partie 2 - probes | 7 |
| Probe Request Evil Twin Attack | 7 |
| Question 1 | 8 |
| Question 2 | 8 |
| Détection de clients et réseaux | 8 |
| Bonus - Hidden SSID Reveal | 9 |
| Question 1 | 10 |

Partie 1 - beacons, authentification

Deauthentication attack

Question 1

Quel code est utilisé par aircrack pour déauthentifier un client 802.11. Quelle est son interprétation ?

De base, selon nos manipulations et analyses, le code utilisé est le 7 - `Class 3 frame received from nonassociated station`.

Cela peut être utilisé pour des raisons légitimes, par exemple pour forcer une station à se connecter à un AP plus proche dans le cas où une STA reçoit une trame de données ou de gestion d'une station qui n'est pas connectée à l'AP.

Question 2

a) A l'aide d'un filtre d'affichage, essayer de trouver d'autres trames de déauthentification dans votre capture. Avez-vous en trouvé d'autres ? Si oui, quel code contient-elle et quelle est son interprétation ?

Nous avons utilisé le filtre `(wlan.fc.type == 0)&& (wlan.fc.type_subtype == 0x0c)` pour trouver les trames de déauthentification.

Ce filtre Wireshark capture les trames Wi-Fi dont le type est de 0 (Management frames) et dont le sous-type est 0x0c (Beacon frame).

On a effectivement trouvé d'autres trames de déauthentification, par exemple la suivante :

| No. | Time | Source | Destination | Protocol | Leng | Info |
|--|-------------|----------------------------------|----------------|----------|------|-------------------|
| 2052 | 4.192465606 | Cisco_1e:ce:00 | Broadcast | 802.11 | 39 | Deauthentication, |
| 2053 | 4.193216189 | Cisco_1e:ce:00 | Broadcast | 802.11 | 38 | Deauthentication, |
| 2054 | 4.193614864 | Cisco_1e:ce:00 | Broadcast | 802.11 | 39 | Deauthentication, |
| 2055 | 4.193644292 | MS-NLB-PhysServer-24_3a:c8:2a:0f | Cisco_1e:ce:00 | 802.11 | 44 | Deauthentication, |
| 2056 | 4.194905006 | Cisco_1e:ce:00 | Broadcast | 802.11 | 39 | Deauthentication, |
| 2057 | 4.195455318 | Cisco_1e:ce:00 | Broadcast | 802.11 | 38 | Deauthentication, |
| 2058 | 4.197645512 | Cisco_1e:ce:00 | Broadcast | 802.11 | 38 | Deauthentication, |
| 2059 | 4.198230402 | Cisco_1e:ce:00 | Broadcast | 802.11 | 38 | Deauthentication, |
| ▼ Fixed parameters (2 bytes) | | | | | | |
| Reason code: Deauthenticated because sending STA is leaving (or has left) IBSS or ESS (0x0003) | | | | | | |

FIGURE 1 – Deauth aircrack other random

qui contient le code 3 - `Deauthenticated because sending STA is leaving IBSS or ESS`.

Ici c'est probablement une station qui quitte le réseau local sans fil (WLAN) auquel elle était connectée. et qui notifie l'AP de son départ du réseau.

b) Développer un script en Python/Scapy capable de générer et envoyer des trames de déauthentification. Le script donne le choix entre des Reason codes différents (liste ci-après) et doit pouvoir déduire si le message doit être envoyé à la STA ou à l'AP :

Le script est disponible [ici](#)

Question 3

Quels codes/raisons justifient l'envoi de la trame à la STA cible et pourquoi ?

Le code 1 : Car il ne spécifie pas la raison.

Le code 4 : La STA est inactive depuis un certain temps et elle doit donc être déconnectée.

Le code 5 : Envoyée à la STA car l'AP est surchargé et il ne peut pas associer de stations supplémentaires.

Question 4

Quels codes/raisons justifient l'envoi de la trame à l'AP et pourquoi ?

Le code 1 : Car il ne spécifie pas la raison.

Le code 8 : Il est envoyé à l'AP car la STA quitte le réseau (BSS). Il est utilisé dans le cas où le réseau est surchargé et que l'AP doit déconnecter des clients et les rediriger vers un autre AP (load-balancing).

Question 5

Comment essayer de déauthentifier toutes les STA ?

En envoyant une trame de deauth **Broadcast** (FF:FF:FF:FF:FF:FF) à l'adresse de l'AP.

Question 6

Quelle est la différence entre le code 3 et le code 8 de la liste ?

Comme dit précédemment le code 8 est plutôt utilisé lorsque le réseau est chargé de clients et donc l'AP va déconnecter certains clients et les rediriger vers un autre AP. Le code 3 ne propose pas de redirection vers un autre AP, il est plutôt utilisé lors d'une violation de sécurité.

Question 7***Expliquer l'effet de cette attaque sur la cible***

On va pouvoir forcer la déconnexion de clients qui se trouvent sur un AP, on va pouvoir le faire en forgeant une trame de deauthenticatation avec l'adresse MAC de la victime. De plus on pourra spécifier une raison à cette déconnection afin de la faire passer pour "normal".

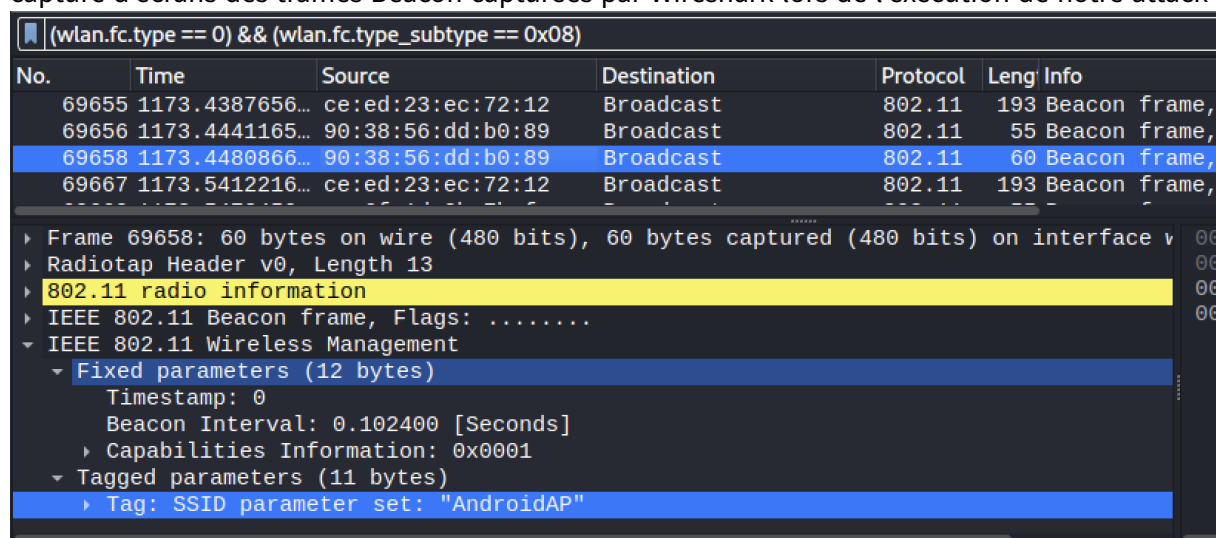
Fake channel evil tween attack**Question 1*****Expliquer l'effet de cette attaque sur la cible***

Cette attaque permet d'envoyer des beacons forgés afin de simuler un faux réseau. Ces trames forgées sont créées en récupérant les informations d'un des réseaux environnant. La victime pensera donc qu'il s'agit d'un vrai réseau sur lequel il peut se connecter. Ensuite l'attaquant surveiller le trafic et voler des informations des différentes victimes.

Dans notre cas le script permet de faire cette attaque sur un réseau ouvert (ce qui n'est pas très réaliste). De plus nous n'en faisons rien, dans le sens où nous ne faisons pas d'actions malveillantes sur les personnes qui se connecte sur notre faux AP.

Le script est disponible [ici](#), il est "interactif" et donc ne prends pas de paramètres en particulier.

Nous avons fait quelques tests avec un partage de connexion Android (AndroidAP), voici une capture d'écrans des trames Beacon capturées par Wireshark lors de l'exécution de notre attack :



SSID flood attack

Développer un script en Python/Scapy capable d'inonder la salle avec des SSID dont le nom correspond à une liste contenue dans un fichier text fournit par un utilisateur. Si l'utilisateur ne possède pas une liste, il peut spécifier le nombre d'AP à générer. Dans ce cas, les SSID seront générés de manière aléatoire.

Le script est disponible [ici](#). On peut soit passer un fichier txt contenant un nom par ligne, soit utiliser le script sans liste. A ce moment l'utilisateur doit entrer un nombre d'AP à créer, les noms et adresses MAC sont générés aléatoirement à l'aide de la librairie Faker de python (<https://faker.readthedocs.io/en/master/#basic-usage>)

Exemple de fonctionnement avec une liste fournie par l'utilisateur.

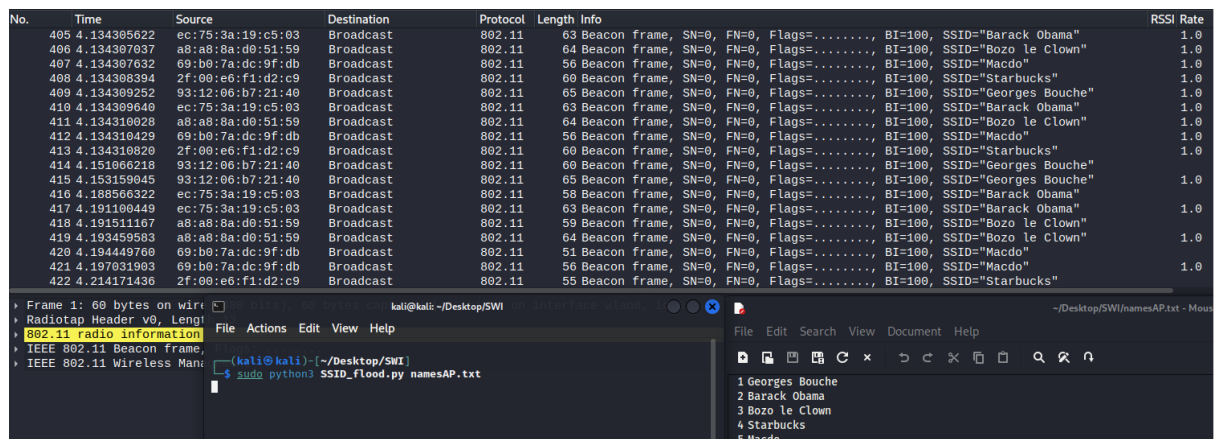


FIGURE 2 – SSID flooding avec liste fournie par l'utilisateur

Exemple de fonctionnement sans liste de noms fournie par l'utilisateur.

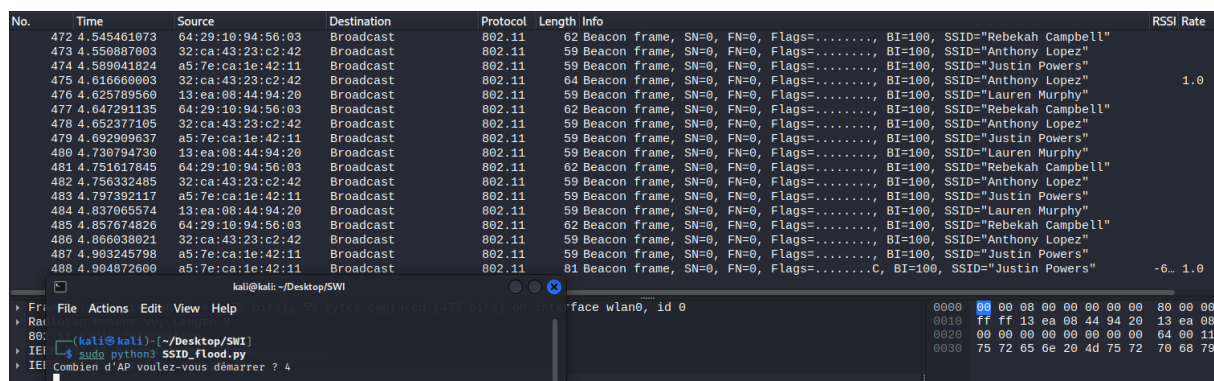


FIGURE 3 – SSID flooding sans liste fournie par l'utilisateur

Partie 2 - probes

Probe Request Evil Twin Attack

Développer un script en Python/Scapy capable de detecter une STA cherchant un SSID particulier - proposer un evil twin si le SSID est trouvé (i.e. McDonalds, Starbucks, etc.).

Exemple du fonctionnement du script (disponible [ici](#)).

Notes: l'interface pour sniffer les paquets est "wlan0". Il peut être nécessaire de la changer dans le script dans le cas où l'interface serait wlan0mon.

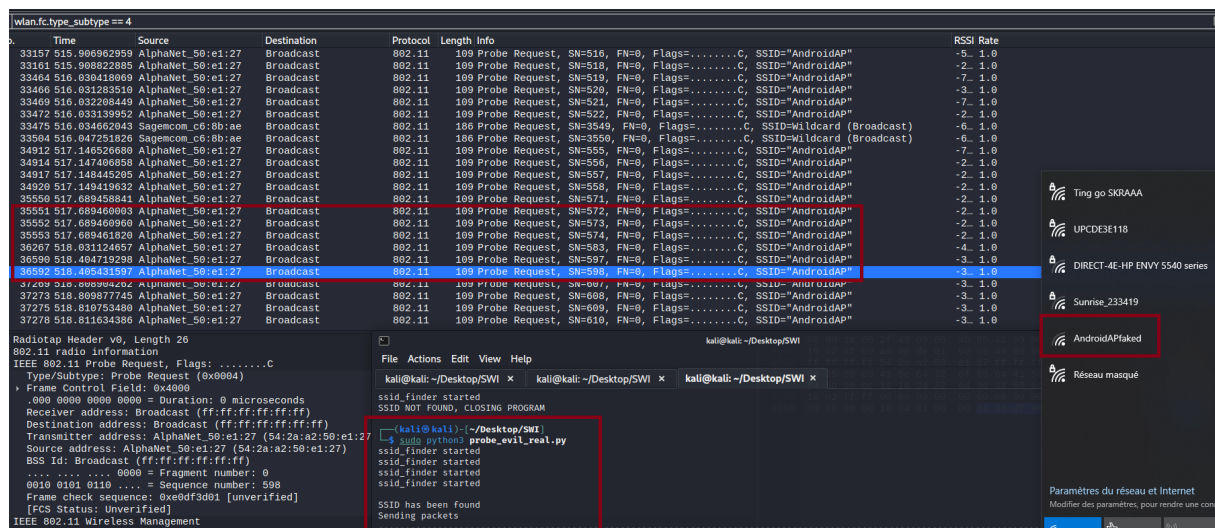


FIGURE 4 – Exemple du fonctionnement du script

Afin de montrer le bon fonctionnement du script, le “faux réseau” créé lors de la détection de probe-request cherchant le SSID “AndroidAP” est nommé AndroidAPfaked. Il est évident que dans le cas d’une réelle attaque, le nom du “faux réseau” doit être le même que le vrai. On voit sur la capture d’écran que des Probe Requests sont envoyées à la recherche du SSID “AndroidAP”. A ce moment, on peut voir que le faux réseau (AndroidAPfaked) est créé à droite dans la liste des wifis détectés par une machine Windows.

Question 1

Comment ça se fait que ces trames puissent être lues par tout le monde ? Ne serait-il pas plus judicieux de les chiffrer ?

Ces trames doivent être envoyées en clair car elles sont utilisées par les clients afin de rechercher activement des réseaux. Si les trames probes étaient chiffrées, les points d'accès ne pourraient pas lire le contenu et ne pourraient donc répondre avec des probes responses.

Question 2

Pourquoi les dispositifs iOS et Android récents ne peuvent-ils plus être tracés avec cette méthode ?

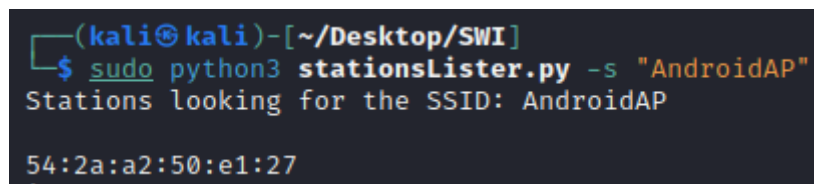
Les appareils iOS et Android récents utilisent des adresses MAC aléatoires pour les trames probes. Par conséquent on aurait beaucoup plus de peine à les tracer puisqu'on ne pourrait plus comparer les adresses MAC.

Détection de clients et réseaux

Développer un script en Python/Scapy capable de lister toutes les STA qui cherchent activement un SSID donné

Exemple du fonctionnement du script (disponible [ici](#)).

Notes: l'interface pour sniffer les paquets est "wlan0". Il peut être nécessaire de la changer dans le script dans le cas où l'interface serait wlan0mon.



```
(kali㉿kali)-[~/Desktop/SWI]
$ sudo python3 stationsLister.py -s "AndroidAP"
Stations looking for the SSID: AndroidAP

54:2a:a2:50:e1:27
```

FIGURE 5 – Exemple du fonctionnement du script

Le script liste simplement les stations qui recherchent le SSID "AndroidAP".

Développer un script en Python/Scapy capable de générer une liste d'AP visibles dans la salle et de STA détectés et déterminer quelle STA est associée à quel AP.

Exemple du fonctionnement du script (disponible [ici](#)).

Notes: l'interface pour sniffer les paquets est "wlan0". Il peut être nécessaire de la changer dans le script dans le cas où l'interface serait wlan0mon.

```
(kali@kali)-[~/Desktop/SWI]
$ sudo python3 displayLinkedSTaToAP.py

STAs                                     APs
dc:4f:22:18:00:e8                       24:79:2a:b3:ac:78
01:00:5e:7f:ff:fa                       1c:d6:be:75:e6:51
01:00:5e:00:00:07                       fc:f1:36:8a:10:85
68:c6:3a:95:28:d9                       00:0c:29:9a:ce:9a
01:00:5e:00:00:16                       c6:0d:79:d2:21:77
01:00:5e:00:00:fb                       c6:0d:79:d2:21:77
33:33:00:00:00:fb                       c6:0d:79:d2:21:77
01:80:c2:00:00:13                       08:02:8e:8d:a2:02
80:9f:9b:e8:4c:4e                       0c:8e:29:fd:08:0c
01:80:c2:00:00:13                       08:02:8e:8d:53:d8
00:0e:00:03:31:5e                       b4:79:c8:82:df:e8
00:0e:00:0c:95:cc                       24:79:2a:b3:ac:78
^C
```

FIGURE 6 – Exemple du fonctionnement du script

Le script liste les stations et l'AP auxquelles ces dernières sont connectées.

Bonus - Hidden SSID Reveal

Développer un script en Python/Scapy capable de reveler le SSID correspondant à un réseau configuré comme étant "invisible".

Script disponible [ici](#).

Question 1

Expliquer en quelques mots la solution que vous avez trouvée pour ce problème ?

Basiquement, quand un AP est configuré pour ne pas diffuser son SSID il émet des beacons avec un SSID vide. On peut donc dans un premier temps monitorer le trafic sans-fils, capturer les beacons qui ont ce champ SSID vide. On récupère alors l'adresse MAC de l'AP. Ensuite on authentifie les clients qui sont connectés à cet AP. Les clients vont alors essayer de se reconnecter à l'AP. Lors de cette reconnection, les STA vont envoyer des probe requests avec le SSID de l'AP. On peut donc capturer ces trames et récupérer le SSID de l'AP.