Evil twin attack on WPA2-Enterprise networks.

Project for the Télécom SudParis course NET4104.

Nicolas Rocq, Benoit Marzelleau, Baptiste Sauvé,
Baptiste Legros, Tom Burellier

2024-05-06

Contents

List of Figures

1 Introduction

Ce projet vise à exploiter une faiblesse de configuration rencontrée dans la majorité des réseaux WPA2-Entreprise. En effet, il est très courant que les clients n'aient pas mis en oeuvre la validation du certificat CA, ce qui permet à un tier d'usurper l'identité d'un point d'accès Wi-Fi utilisant le mécanisme de sécurité WPA2-Entreprise.

En particulier, on s'intéressera aux réseaux utilisant le protocole d'authentification PEAP-MSCHAPv2 pour sa popularité. En l'espèce, on montrera dans quel mesure la faiblesse de configuration susmentionnée permet d'obtenir l'empreinte MD4 du mot de passe de l'utilisateur.

1.1 Contributeurs

Encadrant :

- Rémy Grünblatt[1]

Étudiants :

- Nicolas Rocq[2]
- Benoit Marzelleau[3]
- Baptiste Sauvé[4]
- Baptiste Legros[5]
- Tom Burellier[6]

[1]https://github.com/rgrunbla
[2]https://github.com/Nishogi
[3]https://github.com/xanode
[4]https://github.com/Nepthales
[5]https://github.com/Direshaw
[6]https://github.com/Balmine

2 EAP et PEAP

Le protocole EAP (*Extensible Authentification Protocol*) est un protocole de communication réseau qui est utilisé pour authentifier un partenaire.

PEAP (*Protected Extensible Authentication Protocol*) est un protocole utilisé pour protéger les réseaux Wi-Fi. PEAP est une extension de EAP (*Extensible Authentication Protocol*), qui encapsule les messages EAP dans un tunnel TLS afin d'améliorer la sécurité des échanges.

EAP est un protocole d'authentification dévelopé initialement pour les réseau filaire supportant les connexions point-à-point, où la sécurité physique était présupposée. EAP a ensuite été adopté pour l'authentification 802.1x pour le contrôle d'accès des réseaux locaux, mais il est vite apparût que EAP n'était pas suffisamment sécurisé dans un ensemble de cas d'utilisation.

PEAP a été développé pour pallier à ces faiblesses. En l'espèce, PEAP encapsule la session EAP dans un tunnel TLS, de façon à protéger les échanges entre le client et le serveur d'authentification, qui, entres autres, sont susceptibles de contenir les informations d'identification du client.

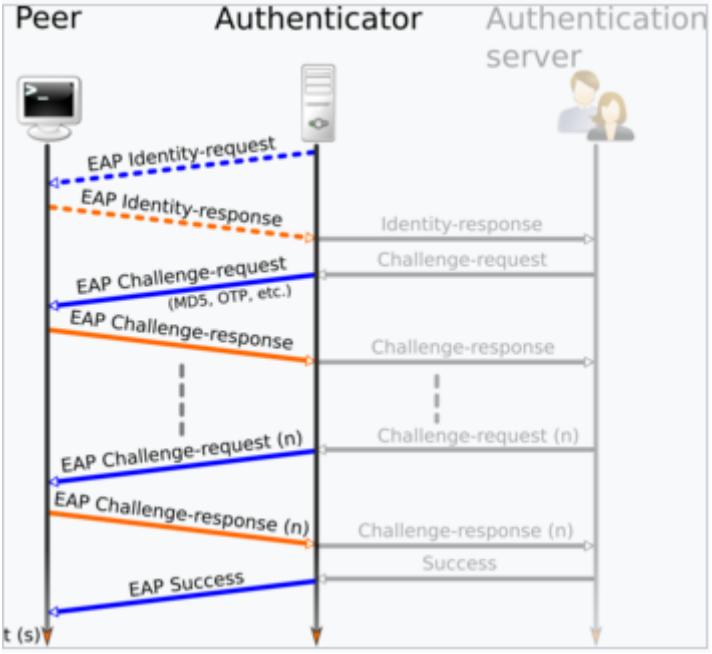

Figure 1: Séquence EAP

Un réseau Wi-Fi utilisant 802.1X est composé de trois éléments principaux : - Le *serveur* est le composant central d'un réseau à accès contrôlé. Il gère les fonctions d'authentification et d'autorisation des supplicants. Situés dans le réseau de confiance, c'est lui qui décide si la connexion d'un supplicant au réseau à accès contrôlé est autorisée ou refusée. - le *client* est le composant qui contrôle l'accès au réseau. Il fait le lien entre les supplicants et le serveur d'authentification. - le (ou les) *supplicant* est le composant qui demande l'accès au réseau.

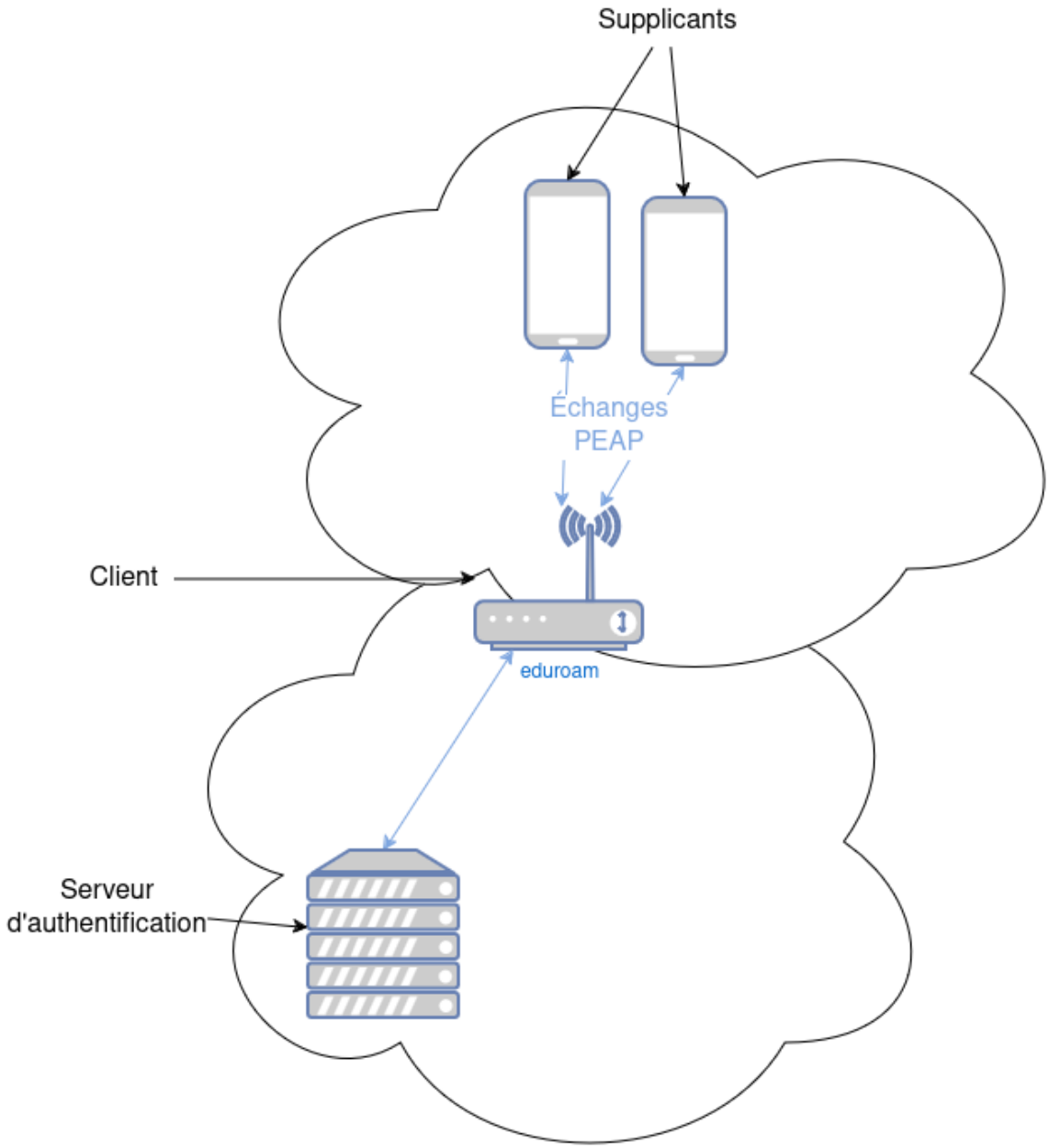

Figure 2: Architecture 802.1X

La connexion à un tel réseau se fait en plusieurs étapes (méthode EAP) : 1. **Initialisation** : le client détecte la tentative de connexion du supplicant au réseau. 2. **Identification** : - Le client transmet au supplicant la demande d'identification ; - le supplicant retourne au client son identité ; - le client transmet l'identité du supplicant au serveur. 3. **Négociation EAP** : - Le serveur indique au client la méthode d'authentification à utiliser (dans notre cas d'utilisation, MSCHAPv2) ; - le client transmet cette information au supplicant ; - si le supplicant accepte la méthode d'authentification, il procède à l'étape d'authentification, sinon il envoie au client les méthodes supportées et l'étape de négociation recommence. 4. **Authentification** : - le serveur et le supplicant échangent des messages dont la nature est tributaire de la méthode d'authentification utilisée (voir la section sur MSCHAPv2 pour nore cas d'utilisation) ; - le serveur indique au client si l'authentification a réussi ou échoué. Le supplicant est connecté au réseau selon le succès de celle-ci.

Le protocole PEAP a la particularité de de procèder à l'authentification du supplicant dans un tunnel TLS au moyen d'une méthode EAP. Cette étape d'authentification précédée d'une phase où le serveur s'authentifie aurpès du supplicant au moyen d'un certificat. Or, la (grande) majorité des réseaux WPA2-Entreprise n'imposent pas aux utilisateurs de valider le certificat du serveur (souvent pour des raisons de simplicité, jugeant que l'import par l'utilisateur d'un certificat sur son équipement est trop délicat pour un public non initié), ce qui rend possible une attaque de type « jumeau maléfique », dans laquelle un attaquant peut se faire passer pour un point d'accès légitime et intercepter les informations d'identification du client.

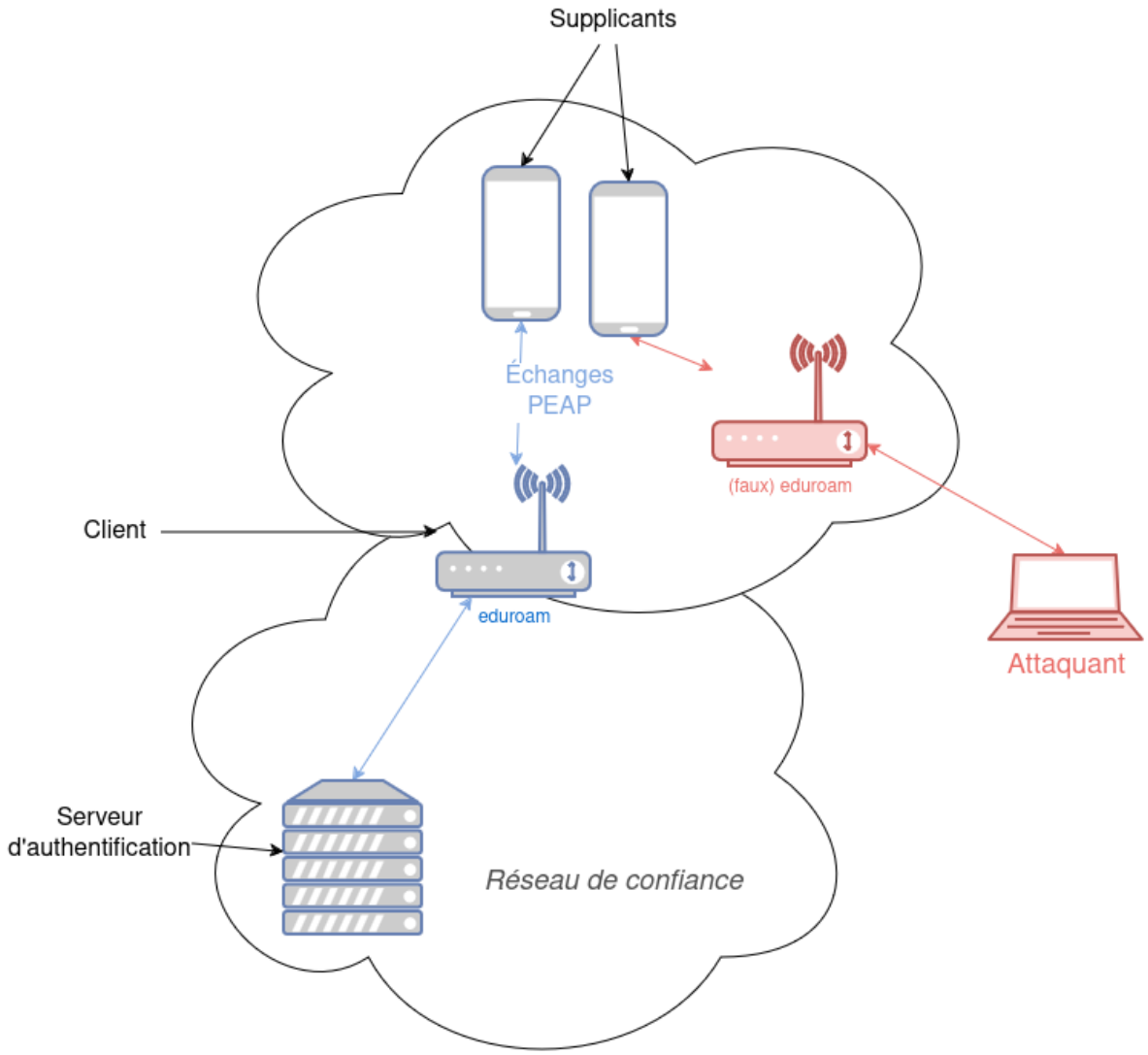

Figure 3: Attaque de type "jumeau maléfique"

3 MSCHAPv2

MSCHAPv2 est une méthode d'authentification de type CHAP (*Challenge Handshake Authentication Protocol*). L'objectif de cette classe de protocoles est de permettre à un client de s'authentifier en respectant les critères suivants : - pas d'échange en clair du mot de passe ou d'une empreinte de celui-ci ; - rejouabilité des échanges de nul effet pour un tier qui l'aurait intercepté.

Pour y parvenir, les protocoles de type CHAP réclament une preuve d'identité du client en lui demandant de répondre à un défi qui ne peut être relevé que par une entité connaissant le mot de passe. Par exemple, il pourrait être demandé au client de chiffrer un nombre aléatoire (fourni par l'authentificateur) avec le mot de passe (chose qui ne peut être réalisée qu'en connaissant le mot de passe).

Le protocole MSCHAPv2 fonctionne de la manière suivante :

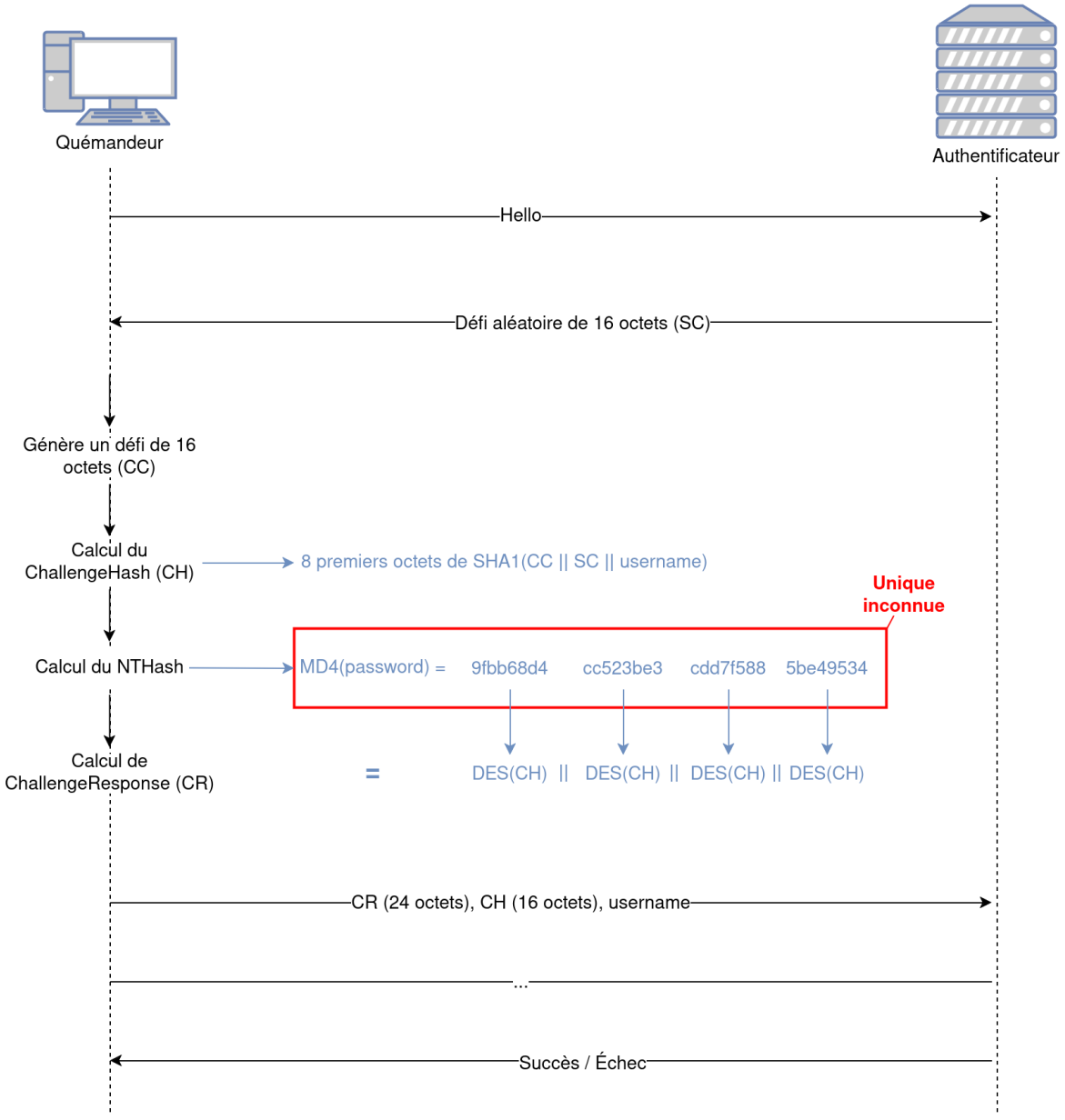

Figure 4: Séquence MSCHAPv2

1. Le serveur d'authentification envoie 16 octets aléatoires au client (le défi, *SC*) ;
2. Le client, pour prouver son identité, réalise :
    - Génération d'un défi de 16 octets (*CC*) ;
    - Calcul du `ChallengeHash` (*CH*) : `SHA1(CC || SC || username)` ;
    - Calcul de l'empreinte MD4 du mot de passe ;
    - Calcul de la réponse au défi à partir de l'empreinte du défi ;
3. Le client envoie au serveur le nom d'utilisateur et la réponse au défi ;
4. Le serveur vérifie la réponse au défi en vérifiant que la réponse du client est correcte, et accepte ou non l'authentification.

Ainsi la seule inconnue pour un attaquant qui intercepte les échanges est l'empreinte MD4 du mot de passe de l'utilisateur, qui est utilisé pour construire les trois clés DES utilisées pour calculer la réponse au défi. Tout autre élément du protocole est soit envoyé en clair, soit facilement déductible des échanges.

4 L'attaque

4.1 Evil twin

4.1.1 Principe

Une attaque de type "evil twin" exploite la façon dont les clients WiFi reconnaissent les réseaux, en se basant principalement sur le nom du réseau (ESSID) sans exiger de la station de base (point d'accès) qu'elle s'authentifie auprès du client. Il s'agit d'une faiblesse de configuration rencontrée dans la majorité des réseaux WPA2-Entreprise, lesquels n'imposent pas aux utilisateurs de contrôler le certificat présenté par le point d'accès pour des raisons de simplicité.

Les points clés sont les suivants de l'attaque sont les suivants :

- **Différenciation difficile** : Il est difficile de différencier un point d'accès légitime d'un point d'accès malveillant lorsque leurs ESSID sont confondus et qu'ils partagent le même mécanisme de sécurité (WPA2-Enterprise dans notre cas). D'autant plus que les réseaux WPA2-Enterprise que l'ont retrouve dans les établissements utilisent souvent plusieurs point d'accès avec le même ESSID pour étendre la courverture de manière transparente pour les utilisateurs finaux.
- **Itinérance des clients et manipulation des connexions** : Le protocole 802.11 permet aux appareils de passer d'un point d'accès à l'autre au sein d'un même ESS. Il est possible d'exploiter cette possibilité en incitant un appareil à se déconnecter de son point d'accès actuel et à se connecter à un point d'accès malveillant. Il est possible d'y parvenir en offrant un signal plus fort ou en perturbant la connexion au point d'accès légitime en envoyant des paquets de désauthentification ou en le brouillant.

4.1.2 Mise en œuvre

Nous allons configurer le point d'accès malveillant à l'aide de `hostapd-wpe` qui est un correctif de `hostapd` qui permet de réaliser l'attaque "evil twin" et surtout d'obtenir les informations d'identification du client (dont le sujet est traité ci-après) échangés lors de l'authentification (et normalement inaccessibles avec `hostapd` seul).

```
# Installation de hostapd-wpe
sudo apt install hostapd-wpe
```

Nous configurons `hostapd-wpe` de la sorte pour qu'il diffuse un point d'accès avec le même ESSID que le réseau cible.

```
interface=wlan0
ssid=eduroam
channel=1
ignore_broadcast_ssid=0
eap_user_file=mi-net4104/hostapd-wpe.eap_user
ca_cert=mi-net4104/attack/ca.pem
server_cert=mi-net4104/attack/server.pem
private_key=mi-net4104/attack/server.pem
private_key_passwd=password
dh_file=mi-net4104/attack/dh
eap_fast_a_id=101112131415161718191a1b1c1d1e1f
eap_server=1
```

```
eap_fast_a_id_info=hostapd-wpe
eap_fast_prov=3
ieee8021x=1
pac_key_lifetime=604800
pac_key_refresh_time=86400
pac_opaque_encr_key=000102030405060708090a0b0c0d0e0f
wpa=2
wpa_key_mgmt=WPA-EAP
wpa_pairwise=TKIP CCMP
```

```
# Lance le point d'accès malveillant
sudo hostapd-wpe hostapd-wpe.conf -s
```

Voici les logs de hostapd-wpe lorsqu'un client se connecte

Figure 5: Logs de Hostapd-wpe

Comme on peut le voir sur le logs, on a hostpad-wpe qui nous donne directement ce que nous devons donner à haschat pour retrouver le mots de passe utilisé.

Ensuite la manière simple de récupéré le mots de passe si le mots de passe du client est simple est d'utilisé haschat avec une attaque par dictionnaire en utilisant la base de mots de passe la plus connu : rockyou.txt

```
# Lance le point d'accès malveillant
hashcat64.exe -m 5500 -a 0 <Le challenge md4 que nous donne hostapd-wpe>
    rockyou.txt
```

Si le mots de passe est simple il est souvent dans la base de mots de passe de rockyou.txt et donc on peut le retrouver facilement.

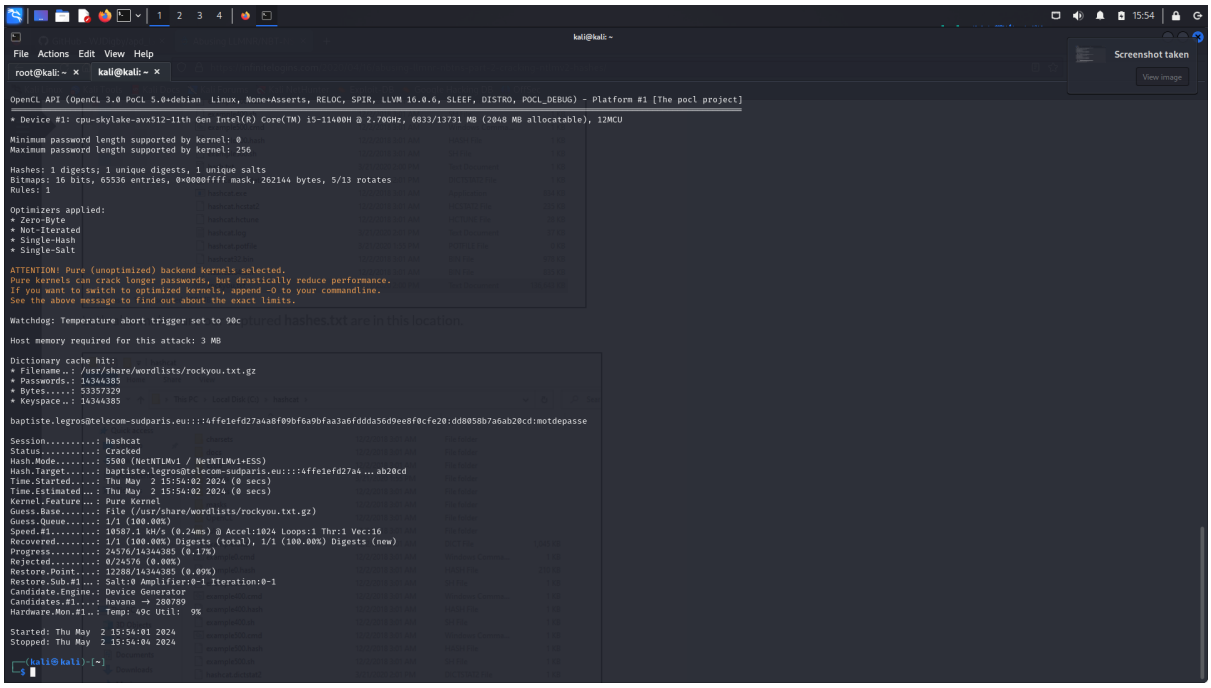

Figure 6: Logs de hashcat

4.2 MSCHAPv2

Maintenant que nous avons obtenu les informations d'identification du client, nous allons nous intéresser à la manière dont nous pouvons les utiliser pour obtenir le mot de passe de l'utilisateur et nous connecter au réseau à l'aide de ces informations. Nous pouvons nous concentrer auprès de la méthode d'authentification (MSCHAPv2), et remarquer que l'empreinte MD4 du mot de passe de l'utilisateur est suffisante pour s'authentifier en ce qu'elle agit comme le mot de passe lui-même (cf. section au sujet de MSCHAPv2).

4.2.1 Attaque par dictionnaire ?

Une première approche pour obtenir le mot de passe de l'utilisateur pourrait être de réaliser une attaque par dictionnaire sur l'empreinte MD4. Par exemple, on pourrait simplement calculer l'empreinte MD4 d'un grand nombre de mots de passe possibles, s'en servir pour calculer la réponse à un défi et comparer avec la réponse fournie par le client.

Le problème de cette approche est que la réussite de cette attaque n'est pas garantie, car le mot de passe de l'utilisateur peut être complexe et ne pas figurer dans le dictionnaire.

4.2.2 Attaque par force brute ?

Dans la mesure où le mot de passe de l'utilisateur est susceptible d'avoir une longueur arbitraire et d'être composé de caractères d'un large ensemble, il pourrait être intéressant d'attaquer par la force brute l'empreinte MD4 du mot de passe elle-même. Mais cette empreinte est de 128 bits, soit 2^{128} possibilités, ce qui est bien trop grand pour être réalisable en un temps raisonnable.

4.2.3 Diviser pour mieux régner

L'empreinte MD4 que nous essayons d'obtenir est utilisée comme la clef de trois chiffrements DES. Les clés DES étant de 7 octets, chaque opération DES utilise un morceau de 7 octets de l'empreinte MD4. Ainsi, au lieu de chercher l'empreinte MD4 elle-même, on pourrait chercher les trois clés DES qui permettent de la construire. Dès lors, plus besoin d'attaquer par la force brute une empreinte de 128 bits, mais trois clés de 56 bits chacune.

Comme il y a 3 opérations DES et que ces opérations sont indépendantes les unes des autres, on a une complexité globale de $3 \times 2^{56} = 2^{57.59}$, ce qui est bien mieux que 2^{128}.

Mais il y a quelque chose qui ne va pas. En l'espèce, nous avons besoin de trois clefs DES de 56 bits pour un total de 168 bits, alors que l'empreinte MD4 n'en fait que 128 : il manque 40 bits. Lors de la réalisation de MSCHAPv2, Microsoft a palié à ce problème en ajoutant 40 bits nuls à l'empreinte MD4, rendant la troisième clef DES d'une longueur effective de 16 bits.

Ainsi, il n'y a que 2^{16} possibilités pour cette dernière clef DES, ce qui est tout à fait réalisable par la force brute. La complexité totale de l'attaque est donc de $2^{56} + 2^{56} + 2^{16} = 2^{57}$. C'est considérablement mieux.

```
    Clef 1          Clef 2          Clef 3
+--------------+  +------------+   +------------+
|?|?|?|?|?|?|?|  |?|?|?|?|?|?|   |?|?|0|0|0|0|
+--------------+  +------------+   +------------+
```

5 Réalisations et contraintes

5.1 Historique chronologique

5.1.1 Au début : L'ESP32-S3

Pour mettre en place notre solution de jumeau maléfique sur les bornes eduroam, nous avons tout d'abord utilisé le matériel fourni par le professeur, c'est à dire une carte ESP32-S3. Cette carte est un microcontrôleur intégrant la gestion du wifi et du bluetooth. Nous avons donc tout d'abord passé un moment à comprendre comment fonctionne l'ESP 32-S3 et mettre en place nos environnements de travail, à l'aide de la documentation officielle afin d'émettre un réseau wifi simple qui utilisait le protocole WPA2 (personnel). Tout le code de l'environnement de l'ESP32 est disponible sur la page github du projet. Sachant que pour mettre en place l'environement de travail nous avons suivis les instructions de la documentations de l'extension VS code de espressif [https://github.com/espressif/vscode-esp-idf-extension/blob/master/docs/tutorial/install.md].

Une fois l'environnement mis en place et avoir compris comment fonctionnait l'ESP32 nous avons été mis devant la dure réalité que l'ESP 32-S3 ne supportait pas le WPA2 Entreprise en mode access point "ESP32-S3 supports Wi-Fi Enterprise only in station mode." de cette documentation [https://docs.espressif.com/projects/esp-idf/en/stable/esp32s3/api-guides/wifi-security.html], ce qui a résulté en la perte de plusieurs longues heures de travail. La conclusion suite à cet échec fut la suivante, il nous fallait un nouveau point d'accès wifi comme une borne. Après quelques discussions avec le professeur, il a pu nous fournir une borne Cisco flashé sous OpenWRT.

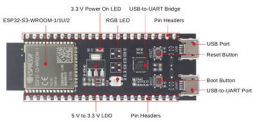

Figure 7: Magnifique carte ESP32-S3

5.1.2 2ème solution : La borne OpenWRT

Après l'échec de l'ESP32, il a donc fallu prendre en main la nouvelle technologie qui est la suivante : Une borne cisco sous OpenWRT.

L'avantage : il existe une magnifique interface HTTP pour gérer l'ensemble des fonctionnalités premières de la borne. L'inconvéniant : c'est quand même vachement plus compliqué de setup une borne Cisco plutôt qu'un ESP32, mais bon avait pas le choix.

Nous avons donc pris du temps pour comprendre le fonctionnement du nouveau matériel et mettre en place notre nouvel environnement de travail. Première étape : il s'agissait de flash notre magnifique nouvelle borne. Je passe l'étape ou nous avons flash la borne 3 ou 4 fois parce que ça marchait pas pour des raisons obscurs, et la fois ou on a flash une snapshot ce qui a donné une image incomplète, sans les drivers wifi, donc impossible d'émettre quoi que ce soit !

Mais je ne vais pas passer l'étape où il a fallu connecter la borne à internet, mais également dans un réseau local pour se ssh pour installer les paquets wpa par défaut et wpa-entreprise. On a tout d'abord essayé un partage de connexion, mais cela ne permettait pas de se ssh. Il a alors fallu à partir de deux interface d'un ordinateur, créer un bridge réseau, l'une des interface connecté à internet via le réseau de la DISI, et l'autre connecté à la borne, ce qui nous a permis de nous ssh en ayant une connexion internet.

Donc finalement, nous avons pu à l'aide de documentation en ligne émettre un réseau wpa-entreprise avec cette borne ! Mais c'est alors que viens le problème suivant : comment est-ce qu'on intercepte les paquets de connexion wifi sans rentrer dans un monde bas niveau dans lequel on a pas envie d'aller ? Et bah il semblerais qu'on peut pas, ou du moins simplement, sans craquer la borne en somme.

Donc, on doit encore écarter cette solution, et en trouver une autre, qui intègre directement l'aspect interception des données, la partie à la limite de l'illégal en somme.

Figure 8: Magnifique borne Cisco Meraki MR42

5.1.3 Solution finale : Hostapd-WPE

Nous voilà à cours de solutions, il nous fallait trouver une alternative pour devenirs de gentils petits hackers très malveillants et intercepter les paquets chiffrés.

Nous sommes tombés un peu au hasard sur le package Hostapd-wpe (big up au moteur de recherche Google).

Ce package contient une version modifiée de hostapd avec le patch hostapd-wpe. Il met en œuvre des attaques d'usurpation d'Authentificateur IEEE 802.1x pour obtenir les informations d'identification des clients. Bah c'est super ! C'est exactement ce dont on a besoin.

Sans rentrer dans les détails, car ils sont dans la partie `03-Attaque`, nous avons configurer le point d'accès malveillant à l'aide de `hostapd-wpe` qui permet de réaliser l'attaque "evil twin" et surtout d'obtenir les informations d'identification du client échangés lors de l'authentification.

Cette solution nous a permis d'arriver à la solution finale de notre projet, l'attaque evil-twin fonctionne sur une raspberry avec l'OS Kali Linux. Ce système d'exploitation a été choisi car ….

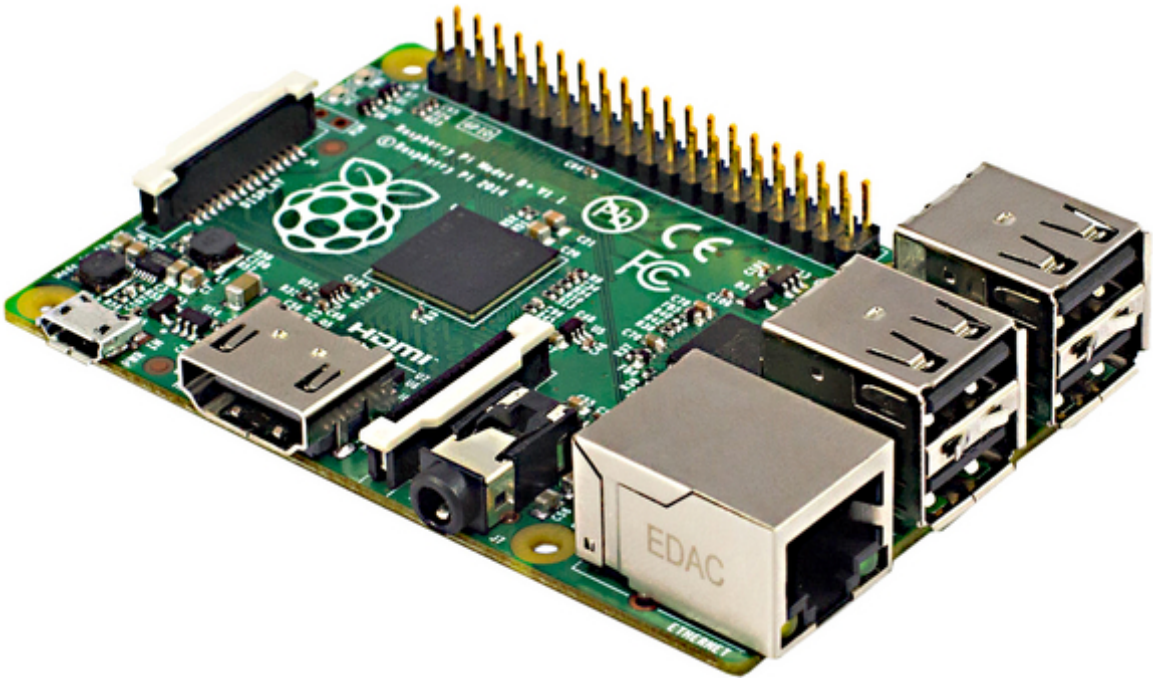

Figure 9: Magnifique Raspberry Pi 1 Model B+ (merci MiNET)

Finalement, nous n'avons pas réussi à flash kali linux sur la raspberry, nous avons donc utilisé une live image de kali linux sur une clé usb, ou nous avons installé hostapd-wpe.

5.2 Réalisations détaillées

5.2.1 Pour l'esp32

Pour l'esp32, la majorité du travail que nous avons pu faire dessus a été de diffuser un SSID spécial :

Disponible au lien `/src/main/esp_wifi.c`.

```
void setup_wifi_ap() {
    wifi_init_config_t cfg = WIFI_INIT_CONFIG_DEFAULT();
    ESP_ERROR_CHECK(esp_wifi_init(&cfg));

    wifi_config_t wifi_config = {
        .ap = {
            .ssid = "Benoiïît mdp = strapontin",
            .ssid_len = strlen("Benoiïît mdp = strapontin"),
            .password = "strapontin",
            .threshold.authmode = WIFI_AUTH_WPA2_ENTERPRISE,

        },
    };

    if (strlen("Your Password") == 0) {
        wifi_config.ap.authmode = WIFI_AUTH_OPEN;
    }

    ESP_ERROR_CHECK(esp_wifi_set_mode(WIFI_MODE_AP));
    ESP_ERROR_CHECK(esp_wifi_set_config(ESP_IF_WIFI_AP, &wifi_config));
    ESP_ERROR_CHECK(esp_wifi_start());
```

```
    printf("AP started with SSID:%s password:%s\n",
           "Benoiiît mdp = strapontin", "strapontin");
}
```

Cette partie a été définie pour mettre en place l'AP wifi, avec un ssid et un mot de passe spécial, comme vous pouvez le voir avec `.threshold.authmode = WIFI_AUTH_WPA2_ENTERPRISE`, nous avons essayé de mettre le mode d'authentification en WPA2_ENTREPRISE, qui est disponible sur les bibliothèques C standarts, mais ne sera pas compatible avec la carte ESP32-S3.

```
static void event_handler(void* arg, esp_event_base_t event_base,
                          int32_t event_id, void* event_data) {
    if (event_base == WIFI_EVENT && event_id == WIFI_EVENT_AP_STACONNECTED
        ) {
        wifi_event_ap_staconnected_t* event = (
            wifi_event_ap_staconnected_t*) event_data;
      char macStr[18];
    sprintf(macStr, ESP_MACSTR, ESP_MAC2STR(event->mac));
        printf("Station %s join, AID=%d\n", macStr, event->aid);
    } else if (event_base == WIFI_EVENT && event_id ==
       WIFI_EVENT_AP_STADISCONNECTED) {
        wifi_event_ap_stadisconnected_t* event = (
            wifi_event_ap_stadisconnected_t*) event_data;
        char macStr[18];
        sprintf(macStr, ESP_MACSTR, ESP_MAC2STR(event->mac));
        printf("Station %s join, AID=%d\n", macStr, event->aid);    }
}
```

Cette méthode est comme son nom l'indique mise en place pour gérer des events, en particulier ici lorsque des clients se connectent et se déconnectent, on affiche dans la console des informations sur le client (MAC, ID d'association du client).

```
void app_main() {
    // Initialize NVS
    esp_err_t ret = nvs_flash_init();
    if (ret == ESP_ERR_NVS_NO_FREE_PAGES || ret ==
       ESP_ERR_NVS_NEW_VERSION_FOUND) {
      // NVS partition was truncated and needs to be erased
      // Retry nvs_flash_init
      ESP_ERROR_CHECK(nvs_flash_erase());
      ret = nvs_flash_init();
    }
    ESP_ERROR_CHECK(ret);
    vTaskDelay(10000 / portTICK_PERIOD_MS); // Delay        printf("
       Station %s join, AID=%d\n", macStr, event->aid);
 for 10000ms (10 seconds)

    ESP_ERROR_CHECK(esp_netif_init());
    ESP_ERROR_CHECK(esp_event_loop_create_default());

    esp_netif_create_default_wifi_ap();

    setup_wifi_ap();

    ESP_ERROR_CHECK(esp_event_handler_register(WIFI_EVENT,ESP_EVENT_ANY_ID
       ,&event_handler, NULL));
}
```

Et enfin, la méthode main, qui va gérer l'initialisation de l'AP, puis gérer les évènements de connexion

et de déconnexion des clients !

6 Annexes

6.1 Génération de pdf avec pandoc

6.1.1 Introduction

Nous avons été amenés à nous pencher sur un moyen de générer un rendu PDF de notre travail qui soit à la fois joli et facile à mettre en place, c'est à dire ne nécessitant pas de compétences particulières en LaTeX

Nous avons donc choisi d'utiliser pandoc qui est un outil de conversion de documents d'un format à un autre. Il est capable de convertir des fichiers markdown en pdf, en utilisant LaTeX pour la mise en page.

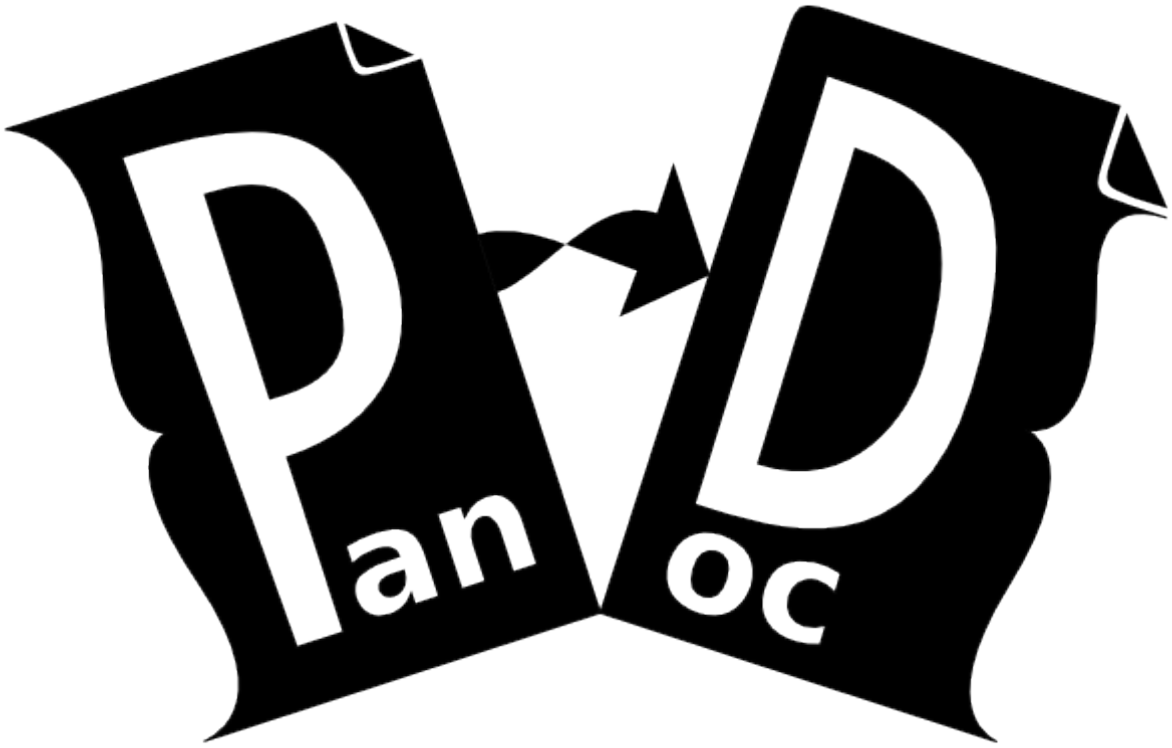

Figure 10: Logo de pandoc

Il est possible de spécifier un template LaTeX pour personnaliser le rendu du pdf. Nous avons choisi d'utiliser le template `eisvogel` qui est un template LaTeX spécialement conçu pour être utilisé avec pandoc. Il est très complet et permet de générer des documents de qualité.

Figure 11: Eisvogel template

6.1.2 Fonctionnement

Tout se passe dans le dossier `docs` qui contient les fichiers markdown à convertir et les fichiers de configuration.

Le dossier `docs` est organisé de la manière suivante :

```
docs├──
 files│
    ├── Eisvogel.png│
    ├── Hashcat.png│
    ├── Hostapd-log.png│
    ├── MSCHAPv2_flow.png│
    ├── Pandoc.png│
    ├── eap.png│
    ├── esp32-S3.jpg│
    ├── meraki-mr42.jpg│
    ├── mitm.png│
    ├── raspberry-pi-1.jpg│
    ├── réseau_802_1x.png│
    ├── réseau_802_1x_evil.png│
    └── tsp.png├──
 md│
    ├── 00-Introduction.md│
    ├── 01-PEAP.md│
    ├── 02-MSCHAPv2.md│
    ├── 03-Attaque.md│
    ├── 04-Réalisation_et_contraintes.md│
    ├── 90-Pandoc.md│
    ├── 99-Sources.md│
    └── HEADER.YAML└──
 pandoc└──
     templates└──
```

```
          eisvogel.tex

5 directories, 22 files
```

- `files` contient les images utilisées dans les fichiers markdown
- `md` contient les fichiers markdown et le fichier `HEADER.YAML` qui contient les métadonnées du document
- `pandoc` contient les fichiers de configuration pour `pandoc`

Chaque modification dans le dossier `md` déclenche la génération du pdf et le résultat est push à la racine du dépôt grâce à un job CI.

Ce fonctionnement permet d'avoir toujours le pdf le plus à jour facilement accessible et est issu de nombreuses itérations pour arriver à un résultat satisfaisant.

7 Sources

7.1 Pandoc

CI with pandoc - https://gitlab.com/pandoc/pandoc-ci-example

7.2 Templating

Templating with pandoc - https://gitlab.cylab.be/a.muls/pandoc-for-pdf