UNIVERSITÀ DEGLI STUDI DI BRESCIA

DIPARTIMENTO DI INGEGNERIA DELL'INFORMAZIONE

Corso di Laurea Magistrale
in Ingegneria Informatica

Tesi di Laurea

EvilRoam: analysis and deployment of evil-twins-based attacks to Eduroam network users' security

Relatore: Chiar.mo Prof. Francesco Gringoli

Laureando:
Gabriele Bellicini
Matricola n. 708588

Anno Accademico 2020/2021

Contents

1 Introduction

Wireless Local Area Networks, usually known as Wi-Fi networks, have quite a long history. Ever since the birth of the 802.11 family of standards, engineers have tried to obtain a high level of security on this kind of networks. Much effort was put into inventing and implementing security mechanisms to protect users from various attacks. Such an effort was necessary because of the nature of WLANs: since data is exchanged over the air, they are intrinsically more insecure than wired networks.

Among the various Wi-Fi families, there is one of them which provides a higher level of privacy, at least up to the $2nd$ generation, with respect to the other ones. This family of networks is usually called WPA-Enterprise, WPA-EAP or simply 802.1X. WPA-Enterprise networks are not as common as WPA-Personal ones, which are used in basically every house nowadays, but are far more common in places where a high number of users has to be connected to the same network.

Between the WPA-Enterprise networks, one stands up for its diffusion across the globe. This network is called "eduroam" and it is supported by a very huge amount of universities in the whole world. As universities usually have thousands of students, hundreds of professors and many other working people, eduroam counts an unimaginable quantity of users. If set up in the right manner, eduroam can provide top-notch security to its users. However, universities as of today still tend to misinform their users on how to properly configure the eduroam network on their smart devices, thus exposing them to various cyber attacks.

Eduroam and 802.1X networks in general have a long history of attacks. Literature is plenty of documents about security flaws in these networks, with some important examples being [1], [2], [3], [4], [5], [6], [7] and [8]. Some of the attacks described in literature surely are well-known, whereas some others are not considered enough or deemed to be not so harmful. Moreover, as we studied some attacks, we realised that many of them, although being known, are strongly lacking of documentation on how to replicate them or to understand them in depth.

Our main goal was to study, develop and test various attacks to eduroam users by means of an evil-twin. An evil-twin is a fully operational device which replicates the behaviours of another one, but with malicious intentions. In our case, the evil-twin had to replicate an eduroam access point, thus letting users authenticate against itself to then perform attacks to users' privacy, stealing various kind of data. Moreover, we deeply concentrated on understanding under which conditions an attack can succeed or is forced to fail.

One of the most important part of our work consisted in deploying three evil-twins inside our university to attack real-world users. In order to do so, we set up three different evil-twins and, after receiving permissions from the ICT team of our university, we installed them in three different places. We then let them run for several days, catching as much data as they could, in order to have better understanding of how many users are in danger under different point of views. The results of the experiment can be seen in a dedicated chapter.

2 Basics of WPA-Enterprise

Nowadays, everyone possessing a smart device is familiar with "Wi-Fi" networks. Wi-Fi networks are omnipresent, as they are in every house, school, bar and restaurant or public place in general. What users refer to as Wi-Fi networks generally are **WPA-Personal** ones, also known as WPA Pre-Shared Key (WPA-PSK) networks, i.e. networks where a user simply needs to know the password set on the access point in order to have access to the internet. **WPA** is the acronym for **Wi-Fi Protected Access**, a set of standards whose purpose is to secure wireless networks. WPA is the heir of Wireless Equivalent Privacy (WEP), a deeply flawed protocol that was marked as insecure shortly after being released.

A least known but equally important set of Wi-Fi networks is the **WPA-Enterprise** one, also called WPA-EAP or 802.1X. Here, in order to authenticate, users don't simply need to know a common password, instead they are provided with some sort of credentials. These credentials typically are a username and a password, which should be unique to each user. WPA-Enterprise networks are far more common than WPA-Personal ones in crowded environments, where a multitude of users needs to have access to the same network. Because of this, WPA-Enterprise networks are really common in enterprises, as the name suggests, and universities, since there can be even thousands of smart devices connected to the same network at the same time. In this kind of places, having a unique secret password is problematic since:

- The password can be easily disclosed to people outside the organization, thus letting them use the network without them having any right;
- If a user leaves the organization, he should also lose the right to use the network, but this requires for the password to be changed on each access point of the organization;
- Changing the password on each access point surely is a time-consuming task,

but most importantly every user needs to reconfigure the network on their smart devices. As there can be thousands of users in the same organizations, this can be quite a big problem.

Moreover, WPA-Personal networks are by design way less secure than WPA-Enterprise ones. For example, it is pretty trivial for a user who has access to a WPA-Personal network to decrypt traffic belonging to other users on the same network. In WPA-Enterprise networks, on the other hand, this is virtually impossible because of how key agreement between access point and smart device is carried out.

Concerning cryptographic aspects, as of WPA2 both WPA-Personal and WPA-Enterprise networks use AES-128 to encrypt traffic between a smart device and an access point. For this reason, brute-forcing a key on both networks is equally hard. In WPA3, which is the latest installment in the WPA world, Personal networks will still have 128-bit encryption keys, while Enterprise ones will use 192-bit keys, thus increasing security against brute-force attacks. However, the $3rd$ generation of WPA claims to increase the strength of WPA-Personal networks too, even if under other aspects, making them comparable to WPA-Enterprise ones.

As it can be imagined, WPA-Enterprise networks are way more architecturally complex than WPA-Personal ones. However, they surely are a must in some environments and, moreover, provide users with a very high degree of security, which is not comparable to WPA-Personal networks, at least as of WPA2.

2.1 Actors in WPA-Enterprise

As said previously, WPA-Enterprise networks are far more complex than WPA-Personal ones. While in the latter there are only access points and smart devices, in the former there are three main actors instead. All of them need to cooperate during the authentication process, but one of them will step back once authentication has ended. The three main actors in WPA-Enterprise networks are the following ones:

- **Authentication Server (AS)**. This server, that usually is a RADIUS server, has access to the database containing users' credentials. It is up to this server to perform authentication of users' devices, telling them whether the authentication was successful or not;

- **Authenticator**. In the wired case, an authenticator can be a switch with Ethernet ports, while in wireless scenarios it is an Access Point (AP). In the latter case, the authenticator communicates on a wireless interface with users' devices, while on a wired interface with the AS. In wireless scenarios, authenticators are also called **Network Access Server (NAS)**. In the initial phase, the authenticator's main role is to relay messages between the AS and the smart device, in order to let them accomplish the authentication run. Once the authentication phase is successfully carried out, the authenticator and the smart device negotiate the symmetric keys to secure the current session. These keys will be used to encrypt and decrypt every message exchanged between the two devices and also to provide their authentication and integrity protection;

- **Supplicant**. What has been referred to as "user's device" or "smart device" until now, is called supplicant in 802.1X terminology. A supplicant is any smart device capable of doing 802.1X authentications. The most common examples are smartphones, laptops and tablets. The supplicant needs to be provided with the user's credentials, in order for it to authenticate itself.

2.2 802.1X, EAP and RADIUS

The authentication process in WPA-Enterprise is based on the **802.1X standard**. According to this standard, a supplicant is assigned to a port of the authenticator that can only forward authentication traffic. Every other kind of traffic is rejected by the authenticator. Once the authentication run is completed and the supplicant is correctly authenticated, the port to which it was assigned is unlocked and every kind of traffic can pass through it. The 802.1X standard was developed for wired connections only (supplicant wired to a switch), but it has been extended in order to be used with wireless LANs too, where an access point replaces the switch.

In order to perform authentication according to the 802.1X standard, the **Extensible Authentication Protocol (EAP)** has been defined within this standard and must be used for authentication runs. EAP, although its name might seem to indicate otherwise, is not an authentication protocol per se, but is, in fact, an authentication framework. To use a particular authentication protocol, what is called an **EAP method** must be agreed and used by the two communicants. Inside the authentication run, the **EAP over LANs (EAPoL)** protocol is used to encapsulate EAP messages inside layer 2 frames exchanged between supplicant and authenticator. This approach results in a really high flexibility, since various authentication methods can be encapsulated in EAP messages, leaving freedom to the system administrators of a specific network on how to authenticate users. More details on EAP methods will be shown in paragraph 3.2.

On the other hand, for what it concerns the communication between the authenticator (or NAS) and the authentication server, other kinds of protocols are used. The most common one is the **RADIUS protocol**, which explains why authentication servers are typically called RADIUS server. This protocol is used to perform authentication, authorization and accounting (AAA).

As it can be seen, while the supplicant and the AS only use a certain protocol (EAPoL and RADIUS respectively), the authenticator must use both of them in order to let the other two communicate. Basically, an authenticator operates as an intermediary between supplicant and AS, translating messages coming from one party to make them comprehensible to the other one.

3 Authentication tools in 802.1X

In WPA2-Enterprise, just like in WPA2-Personal, the authentication process provides mutual authentication: in the former, it is mutual between AS and supplicant, while in the latter it is mutual between access point and supplicant.

In the Enterprise scenario, the AS will first authenticate itself against the supplicant, providing it with a digital certificate, while the supplicant will authenticate itself afterwards, using some kind of credentials, if and only if the AS correctly authenticated previously. It clearly appears that the authentication process can be divided into two macro stages: server-side and supplicant-side authentication.

3.1 Server-side authentication: digital certificates

In 802.1X authentication it is always the AS who authenticates first against the supplicant. In order to do so, the only possibility that the server has is to use a **digital certificate**. Moreover, this certificate is also needed to build the TLS tunnel between supplicant and AS used in tunneled methods. Basics of tunneled authentication methods will be presented in the next subsection.

The main tool used by digital certificates is **public key cryptography**. In public key cryptography, an entity must possess two keys:

- A private key, which must be kept secret by its holder, that provides integrity protection, non repudiation and digital signature;
- A public key, that is disclosed to the world and that can be used by other entities to make sure that their messages can only be read by the entity possessing the private key.

For each private key there exists one and only one public key, so that a message encrypted with a public key can only be decrypted by the corresponding private key and vice versa. Public and private keys are generated for different algorithms and are used by these algorithms in order to give different properties to the exchanged messages, such as confidentiality, integrity protection, non repudiation and digital signature. The most famous public key algorithms surely are **RSA** (Rivest-Shamir-Adleman), **DSA** (Digital Signature Algorithm), its elliptic curve variant (**ECDSA**) and **ElGamal**.

Since the purpose of a public key is to be shared with the rest of the world, it can be included in a digital certificate belonging to the entity that possesses the corresponding private key. A digital certificate contains data about the entity to which it was released, plus the public key. This certificate is then signed with the private key of a trusted party, which is assumed to be signing only legitimate certificates. Upon receiving a digital certificate, another entity will be able to:

1. Determine whether the certificate refers to the right entity by looking at the identification data contained in the certificate;

2. Determine whether the certificate is to be trusted, by validating the signature written by the globally trusted party;

3. Contact confidentially the entity exposing the certificate, using the public key contained in the certificate to encrypt data.

So, in order for the authentication server to authenticate itself against the supplicant, it must be provided with:

- A private key;

- A digital certificate, signed by a **certification authority (CA)**, containing the corresponding public key.

Upon receiving an authentication request, the server provides its certificate to the supplicant, which, in turn, uses the CA's certificate it possesses to verify that that certificate was signed by that specific CA. If the supplicant correctly verifies that the certificate provided by the AS is genuine, server-side authentication is successful and the supplicant can start using the AS public key to exchange messages with it in a confidential way. If, in fact, the supplicant determines that the provided certificate is not valid, then the connection is dropped by the supplicant and server-side authentication does not succeed.

The certificates provided to authentication servers usually are in **X.509** format, which is a really common format in today's web since it is widely used for TLS certificates in HTTPS websites.

An X.509v3 certificate has the following structure:

- **Version number**. This number indicates the version of the X.509 certificate and, as of today, should always be 3 (0x2);
- **Serial number**. This is an increasing number, starting from 1, which is set by the signing CA every time it signs a new certificate. There cannot be two distinct certificates, signed by the same CA, with the same serial number;
- **Signature Algorithm ID**. This field specifies which signature algorithm was used by the CA to sign the certificate. An example for this field is sha256withRSAEncryption, meaning that the signature is generated by hashing data with SHA256 and encrypting the message digest with the CA's RSA private key;
- **Issuer**. This field usually contains pieces of information about the CA that signed the certificate. Examples of these data are country/state and city where the CA resides, the name of the organization, an email address and a DNS common name;
- **Validity Period**. This field is divided into two sub-fields, that are:

  - **Not Before**. A string containing a date and time before which the certificate is not to be considered valid;
  - **Not After**. Similarly to previous one, this field tells when the certificate expires and so becomes invalid;

- **Subject**. Similarly to the "Issuer" field, this one contains pieces of information about the entity who possesses the certificate;
- **Subject Public Key Info**. This field is divided into two sub-fields, containing data about the public key of the subject:
  - **Public Key Algorithm**. It's the name of the asymmetric algorithm that uses the specified public key. An example for this field is "rsaEncryption";
  - **Public Key**. This field contains data regarding the server's public key, i.e. its length and the key itself. In the case of RSA keys, modulus and public exponent are listed here. While the modulus changes between each public key, the exponent is usually equal to 65537 (0x10001), a small but secure value that lets clients verify RSA signatures faster;
- **Certificate signature algorithm**. This field is redundant but must be the same as the "Signature Algorithm ID". It is needed in order to prevent circumvention attacks to the digital signature;
- **Certificate signature**. This is the signature of the certificate, i.e., in the case of
  sha256withRSAEncryption, the digest produced feeding the whole certificate to SHA256, encrypted with the CA's RSA private key.

A CA certificate has basically the same structure shown before, but with some additional fields that indicate it belongs to a CA and not to a server. Moreover, a root

CA certificate has the issuer and subject fields identical, since it has been generated and signed by the CA itself.

To summarize, in order for a supplicant to authenticate an AS, it must receive its certificate and verify that it is not expired. Then, it can hash the content of the certificate and decrypt the signature with the CA's public key: if the two are equal, then it means that the certificate has been signed by that CA and so it can be trusted by the supplicant.

Once server-side authentication has succeeded, the following step in 802.1X authentication requires for the supplicant to authenticate itself against the AS. In this step, the EAP protocol will be used to carry authentication data between the two parties.

3.2 Supplicant-side authentication: EAP methods

As mentioned in the introductory part, authentication in WPA2-Enterprise is carried out according to the IEEE 802.1X standard, which relies on the use of the EAP and EAPoL protocols. During the years, several authentication methods have been developed and can now be used in WPA-EAP networks to carry out an authentication run. In the following list, some of the most used and widely known EAP methods are shown:

- EAP-MD5. This challenge-response method was the first one ratified for EAP by IETF. However, since MD5 is a hash function that is vulnerable to dictionary attacks and does not support the generation of dynamic keys, this EAP method should never be used. It can still be found in various softwares, mainly for historical reasons;

- EAP-PWD (Secure Password). This password-based authentication method requires a shared secret between the AS and the supplicant. This secret is usually a password and can even be a low entropy one, since this method is claimed to be resistant to passive, active and dictionary attacks. It is based on the same mathematical principles as the Diffie-Hellman key exchange protocol, making it a really strong protocol;

- EAP-TLS. With this authentication method, the supplicant is provided with a digital certificate too, containing pieces of information about the user, instead of a password. In order for the AS to authenticate the supplicant, it needs to validate this certificate similarly to how the supplicant validates the AS' one. This EAP method is one of the most important ones as it is one of the safest and doesn't use a password, making it handier for users;

- EAP-TTLS (Tunneled TLS). Unlike EAP-TLS, where the supplicant must have a certificate, in EAP-TTLS a TLS tunnel is built between the AS and the supplicant. Once the tunnel is built, another authentication method can be used inside the tunnel to authenticate the supplicant inside the tunnel, so that everything is protected by TLS. For this reason, EAP-TTLS must always be followed by another authentication protocol, as if used alone it doesn't authenticate the supplicant;

- EAP-PEAP. Just like EAP-TTLS, the purpose of this method is to build a TLS tunnel between AS and supplicant. The difference between the two is that while in EAP-TTLS the exchanged messages contain RADIUS attributes, in EAP-PEAP the attributes follow other rules, specified by the EAP protocol. Basically, the difference between EAP-TTLS and EAP-PEAP only resides in the format of exchanged data.

As it was shown in this list, some methods let a supplicant authenticate in the clear (EAP-PWD and EAP-TLS), while others are only used to build a secure tunnel between the two parties (EAP-TTLS and EAP-PEAP). Inside this tunnel, even insecure authentication methods can be used since the data exchanged are protected by the TLS tunnel. Since in EAP-TTLS and EAP-PEAP the supplicant does not authenticate itself in the first step, these methods are also referred to as phase 1 authentication methods, while the authentication methods used inside the TLS tunnel are called phase 2 authentication methods. As it will be shown in a dedicated section, tunneled methods are the most common ones in eduroam, as they are easier to manage with respect to EAP-TLS (no user certificate management) and they exchange authentication data inside an end-to-end encrypted tunnel, built between the supplicant and the AS of the user's home university.

Various authentication methods can be used inside a TLS tunnel, but there's a major difference between EAP-TTLS and EAP-PEAP: while EAP-PEAP only supports EAP methods (e.g. EAP-MSCHAPv2, EAP-PWD, EAP-SIM, EAP-GTC, etc...), EAP-TTLS supports both EAP and non-EAP methods (such as plain MSCHAPv2, MSCHAPv1, PAP, ...). The main difference between EAP and non-EAP methods is that, as said before, EAP methods carry messages containing EAP attributes, while legacy methods usually carry messages containing RADIUS attributes. Inside of EAP-TTLS, these are some of the authentication methods that can be used:

- Password Authentication Protocol (PAP). This simple protocol requires that the supplicant provides the AS with the user's password. The AS simply needs to check if the password of the user in the database is the same as the one provided. In PAP, the user's password is sent as clear text to the AS, but since it is wrapped in a TLS tunnel, it can only be decrypted by the AS, making TTLS-PAP as strong as TLS;

- Microsoft Challenge Handshake Authentication Protocol (MS-CHAPv1). This protocol, as indicated by the name, was developed by Microsoft and provides one-way authentication. This protocol belongs to the challenge-response family of authentication protocols. In MS-CHAP, the AS sends a challenge to the supplicant, which derives a response based on the password, the username and the challenge. Once received by the AS, it can compute the same quantity and check if they are the same. If so, the supplicant is correctly authenticated;

- MS-CHAPv2. This new, strengthened version of MS-CHAP was still developed by Microsoft and, unlike MS-CHAP, it provides mutual authentication, making it more secure. As of today, it is one of the most used tunneled methods for supplicants' authentication, although being fairly insecure;
- EAP-GTC. This EAP method uses the Generic Token Card protocol developed by CISCO for token cards. As a matter of fact, it is pretty similar to PAP or, at least, it can be configured to mimic PAP in every aspect.

In EAP-PEAP, on the other hand, these are some of the most common authentication methods:

- EAP-MSCHAPv2. It is the same as MSCHAPv2, with the only difference residing in the format of exchanged attributes;
- EAP-GTC. Identical to the one carried inside EAP-TTLS;
- EAP-PWD. The same method discussed previously, but this time it is used inside a TLS tunnel;
- EAP-SIM. This method uses the same authentication scheme used in 2G mobile networks and needs for the AS to be connected to the mobile network;
- EAP-AKA. Similarly to EAP-SIM, this method is the same used for authentication in 3G networks;
- EAP-AKA'. A variant of the previous method, used in mobile networks to allow non-3GPP devices to connect to 3GPP networks.

As it can be seen, there is a pretty huge amount of authentication methods that can be used in 802.1X authentications, some stronger than others. Any organization can then choose the ones it wants to support, thus allowing supplicants to authenticate only if they agree on using one of these methods.

4 Steps of 802.1X authentication

The authentication process is really important in WPA-Enterprise networks and its fundamentals must be explained in order to understand part of the attacks that will be shown later on. In this section, the authentication steps will be shown also using a real life example with packets captured by tcpdump, a Linux command-line tool that lets a user capture packets on network interfaces, and displayed in WireShark, a tool for network packets analysis. The authentication process that is shown in this section undergoes the assumption that either EAP-TTLS or EAP-PEAP is used. The reason behind this is that eduroam, the network we put under attack, generally supports these methods only, so we found it more interesting to discuss this kind of authentications. As seen before, both EAP-TTLS and EAP-PEAP need a phase 2 authentication method (e.g. PAP, MSCHAP, MSCHAPv2, EAP-GTC, ...). The specific phase 2 authentication method, which is the one needed to perform supplicant-side authentication, is irrelevant in this context, as different universities will enforce different phase 2 methods.

Since, in this section, it is assumed that a supplicant is configured to use EAP-TTLS/EAP-PEAP plus a phase 2 authentication method, three ingredients are mandatory in order for a supplicant to perform authentication:

- **Anonymous identity** (also called outer identity). This attribute is used to prevent the supplicant from disclosing the real username with non-trusted parties, since it is sent in the clear and can be easily sniffed. The anonymous identity is not mandatory and supplicants can be configured in order not to use it;

- **Identity** (also called inner identity). This attribute represents the username of the user inside its institution. It usually is something like ⟨name.surname⟩@⟨ institution-name⟩.TLD or ⟨identification-number⟩@⟨institution-name⟩.TLD, so it is strictly tied to the user real identity and, for this reason, should only be exchanged with trusted parties;

- **Password**. A password is needed in methods like PAP, MS-CHAPv2, EAP-PWD and EAP-GTC, which are the most widely used in eduroam. Passwords can have constraints imposed by the institution where the user is enrolled, with the purpose of making some attacks harder to perform.

As mentioned earlier, EAP-PEAP also supports methods such as EAP-SIM, EAP-AKA and EAP-AKA'. These authentication methods are the ones used in 2G and 3G networks and, as such, don't require username or password. However, since eduroam networks generally do not support these methods, they will not be considered any further.

4.1 Anonymous identity exchange

The first step of the 802.1X authentication sees the authentication server asking the supplicant for its identity. This request is, of course, relayed to the supplicant by the NAS. If the supplicant is configured with an anonymous identity, then it will send it to the AS; otherwise, the real identity will be sent out in the clear instead of the anonymous one.

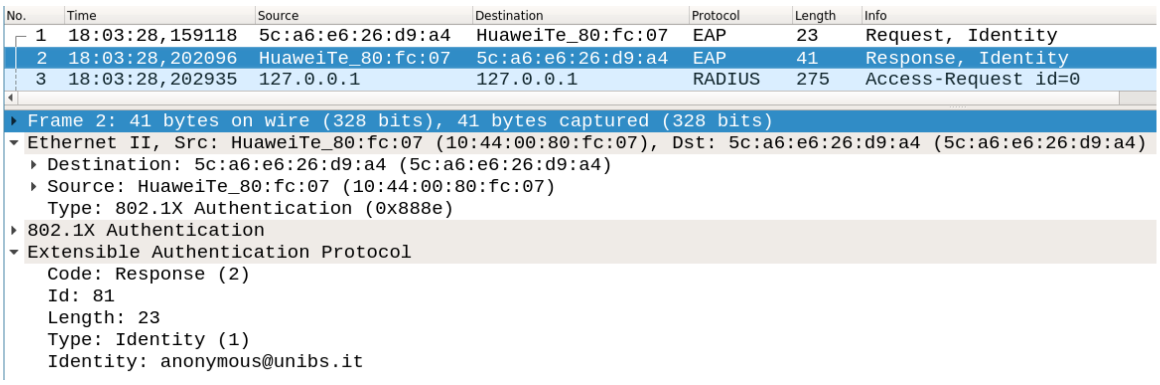
This step always happens in the clear, so anyone within the supplicant and the NAS has access to the outer identity. By using an anonymous identity, a user prevents others from seeing when and where he is trying to authenticate, since the anonymous identity is not related to the user in anyway, but only to the institution.

Upon receiving the outer identity, if the user belongs to that institution, the AS will proceed with server-side authentication, otherwise it will proxy it to the right AS. In either case, an AS must authenticate itself.

4.1.1 Step 1 - example

An example of a real life 802.1X authentication run is presented in the following picture, which is a Wireshark screenshot of packets captured with tcpdump. In this example, the user that wants to authenticate has the following credentials:

- Anonymous Identity = anonymous@unibs.it;
- Identity = m.rossi001@unibs.it;
- Password = password.



```
No.  Time              Source             Destination        Protocol  Length  Info
  1  18:03:28,159118   5c:a6:e6:26:d9:a4  HuaweiTe_80:fc:07  EAP       23      Request, Identity
  2  18:03:28,202096   HuaweiTe_80:fc:07  5c:a6:e6:26:d9:a4  EAP       41      Response, Identity
  3  18:03:28,202935   127.0.0.1          127.0.0.1          RADIUS    275     Access-Request id=0

▸ Frame 2: 41 bytes on wire (328 bits), 41 bytes captured (328 bits)
▾ Ethernet II, Src: HuaweiTe_80:fc:07 (10:44:00:80:fc:07), Dst: 5c:a6:e6:26:d9:a4 (5c:a6:e6:26:d9:a4)
  ▸ Destination: 5c:a6:e6:26:d9:a4 (5c:a6:e6:26:d9:a4)
  ▸ Source: HuaweiTe_80:fc:07 (10:44:00:80:fc:07)
    Type: 802.1X Authentication (0x888e)
▸ 802.1X Authentication
▾ Extensible Authentication Protocol
    Code: Response (2)
    Id: 81
    Length: 23
    Type: Identity (1)
    Identity: anonymous@unibs.it
```

Figure 1: Exchange of the anonymous identity between supplicant and AS

As it can be seen from the picture, the first message is an EAP message sent by the NAS to the supplicant requesting its identity. The supplicant answers with an EAP message with code 0x01, meaning that it is an EAP-Identity message. Its content is, of course, the outer identity, i.e. "anonymous@unibs.it". The content of the message can be partially seen in the lower portion of the image.

4.2 Negotiation of the phase 1 method

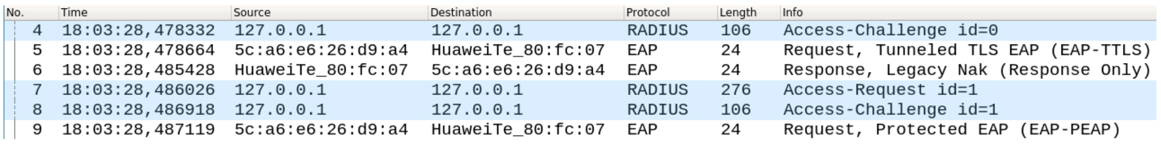
Upon receiving the outer identity of the user, the negotiation of the phase 1 authentication method starts. At this point, the AS sends to the supplicant the method it prefers to use. The supplicant always has the last word in 802.1X, so, upon receiving the phase 1 method advised by the AS, it has two choices:

- If the supplicant has been configured to use the same outer method as the one advised by the AS, the supplicant will agree with the AS and start using that authentication method. This, of course, can only happen if the supplicant physically supports the advised authentication method, which is always the case for both EAP-TTLS and EAP-PEAP;
- In case the supplicant has been configured to use another phase 1 method, then it will send a message to the AS containing an EAP-NAK (Not AcKnowledged, meaning that the supplicant does not want to use the advised method), plus the EAP code of the method that the supplicant wants to use. In this second case, if the method chosen by the supplicant is not supported by the AS, it will answer with another EAP-NAK and the authentication will end with a failure, otherwise the AS will start the selected authentication protocol.

At the end of this stage, both the supplicant and the AS have agreed on which outer authentication method to use, which, in our case, can be either EAP-TTLS or EAP-PEAP.

4.2.1 Step 2 - example

In order to make the example as complete as possible, the authentication run recorded sees the AS proposing to do EAP-TTLS, while the supplicant is configured to do EAP-PEAP.



```
No. Time             Source             Destination        Protocol Length Info
  4 18:03:28,478332  127.0.0.1          127.0.0.1          RADIUS   106    Access-Challenge id=0
  5 18:03:28,478664  5c:a6:e6:26:d9:a4  HuaweiTe_80:fc:07  EAP      24     Request, Tunneled TLS EAP (EAP-TTLS)
  6 18:03:28,485428  HuaweiTe_80:fc:07  5c:a6:e6:26:d9:a4  EAP      24     Response, Legacy Nak (Response Only)
  7 18:03:28,486026  127.0.0.1          127.0.0.1          RADIUS   276    Access-Request id=1
  8 18:03:28,486918  127.0.0.1          127.0.0.1          RADIUS   106    Access-Challenge id=1
  9 18:03:28,487119  5c:a6:e6:26:d9:a4  HuaweiTe_80:fc:07  EAP      24     Request, Protected EAP (EAP-PEAP)
```

Figure 2: Negotiation of the phase 1 authentication method

As it can be seen from the picture, packets 4 and 5 contain the proposal of the AS, which is to do EAP-TTLS. Packets 6 and 7 contain an EAP-NAK and EAP-PEAP, meaning that the supplicant is not willing to do EAP-TTLS and that it wants to do EAP-PEAP instead. Packets 8 and 9 contain a confirmation by the AS that it is willing to do what the supplicant asked for, i.e. EAP-PEAP.

4.3 Phase 1 authentication

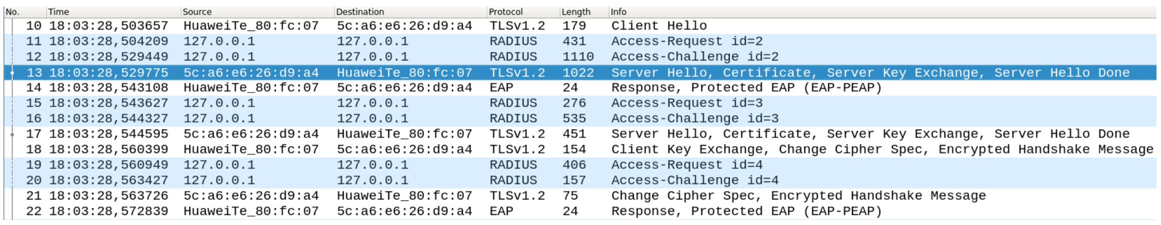
Since both EAP-TTLS and EAP-PEAP need for a TLS tunnel to be built between the supplicant and the AS, this step is the same for both methods. The TLS tunnel setup is carried out according to the latest TLS version supported by both the supplicant and the AS.

In order for this step to succeed, the AS sends its certificate to the supplicant. If the supplicant trusts the certificate, supplicant and authentication server build the TLS tunnel and proceed to agree a symmetric key used to guarantee confidentiality between them.

Once the setup of the TLS tunnel has ended, the two parties will use the TLS protocol to exchange messages, preventing eavesdropping by malicious users. Every message sent inside the TLS tunnel is only visible to the supplicant and the AS.

4.3.1 Step 3 - example

After agreeing to do EAP-PEAP, a TLS tunnel must be established between the AS and the supplicant. For this reason, the TLS protocol must be used at the layer 2 of the TCP/IP stack.



```
No. Time             Source             Destination        Protocol Length Info
 10 18:03:28,503657  HuaweiTe_80:fc:07  5c:a6:e6:26:d9:a4  TLSv1.2  179    Client Hello
 11 18:03:28,504209  127.0.0.1          127.0.0.1          RADIUS   431    Access-Request id=2
 12 18:03:28,529449  127.0.0.1          127.0.0.1          RADIUS   1110   Access-Challenge id=2
 13 18:03:28,529775  5c:a6:e6:26:d9:a4  HuaweiTe_80:fc:07  TLSv1.2  1022   Server Hello, Certificate, Server Key Exchange, Server Hello Done
 14 18:03:28,543108  HuaweiTe_80:fc:07  5c:a6:e6:26:d9:a4  EAP      24     Response, Protected EAP (EAP-PEAP)
 15 18:03:28,543627  127.0.0.1          127.0.0.1          RADIUS   276    Access-Request id=3
 16 18:03:28,544327  127.0.0.1          127.0.0.1          RADIUS   535    Access-Challenge id=3
 17 18:03:28,544595  5c:a6:e6:26:d9:a4  HuaweiTe_80:fc:07  TLSv1.2  451    Server Hello, Certificate, Server Key Exchange, Server Hello Done
 18 18:03:28,560399  HuaweiTe_80:fc:07  5c:a6:e6:26:d9:a4  TLSv1.2  154    Client Key Exchange, Change Cipher Spec, Encrypted Handshake Message
 19 18:03:28,560949  127.0.0.1          127.0.0.1          RADIUS   406    Access-Request id=4
 20 18:03:28,563427  127.0.0.1          127.0.0.1          RADIUS   157    Access-Challenge id=4
 21 18:03:28,563726  5c:a6:e6:26:d9:a4  HuaweiTe_80:fc:07  TLSv1.2  75     Change Cipher Spec, Encrypted Handshake Message
 22 18:03:28,572839  HuaweiTe_80:fc:07  5c:a6:e6:26:d9:a4  EAP      24     Response, Protected EAP (EAP-PEAP)
```

Figure 3: Building of the TLS tunnel

The TLS handshake is not in the scope of this work, so it won't be detailed further. It is shown in this picture, however, that packet 13 contains the certificate of the server, which is a keypoint for the authentication of the AS against the supplicant.

4.4 Inner identity exchange

Once inside the TLS tunnel, the supplicant has already authenticated the AS by means of its certificate, thus being sure that it is talking to a trusted party. At this point, the supplicant is free to disclose its inner identity, since it is end-to-end encrypted with a trusted AS.

Similarly to the first step, the AS requests the inner identity to the supplicant, which will be sent encapsulated in TLS packets. At this point, the AS knows which user wants to authenticate.

4.5 Negotiation of the phase 2 method

Similarly to the second step, in this stage of the authentication process, the AS sends to the supplicant its preferred phase 2 authentication method. Just like in the second step, the supplicant can either agree with the server or disagree with it and specify its own preferred method. As said before, it is always up to the supplicant to choose the authentication method: if the AS supports it, then it must use the one chosen by the supplicant, otherwise the authentication will fail.

4.6 Phase 2 authentication

Once the phase 2 authentication method has been agreed by both parties, the client-side authentication can start. The AS will load the credentials of the user and proceed with the authentication protocol negotiated.

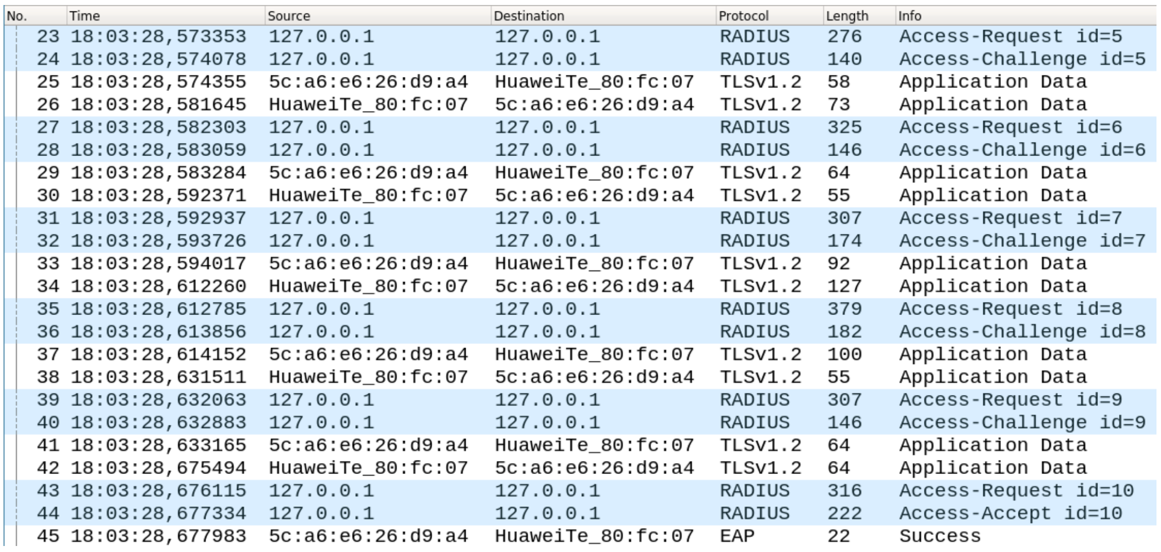
This step only has two possible endings:

- Reject. In case the supplicant failed to authenticate itself, the AS will send an Access-Reject packet. The authenticator will then forbid the supplicant from joining the network;
- Accept. In case the supplicant successfully authenticated, the AS will send an Access-Accept packet, letting both the supplicant and the authenticator know that the supplicant can rightfully join the network.

4.6.1 Steps 4, 5 and 6 - example

What happens after the third step is all encrypted in the TLS tunnel, meaning that its content cannot be trivially retrieved. For this reason, the exchange of the inner identity, the agreement of the phase 2 method and the phase 2 authentication cannot be easily seen in detail in Wireshark.

For this authentication run, the AS was configured to propose EAP-GTC to the supplicant, while the supplicant was configured to do EAP-MSCHAPv2. The second method provides mutual authentication and requires more messages, for this reason the following screenshot contains many packets.



```
No. Time            Source             Destination        Protocol Length Info
 23 18:03:28,573353 127.0.0.1          127.0.0.1          RADIUS   276    Access-Request id=5
 24 18:03:28,574078 127.0.0.1          127.0.0.1          RADIUS   140    Access-Challenge id=5
 25 18:03:28,574355 5c:a6:e6:26:d9:a4  HuaweiTe_80:fc:07  TLSv1.2  58     Application Data
 26 18:03:28,581645 HuaweiTe_80:fc:07  5c:a6:e6:26:d9:a4  TLSv1.2  73     Application Data
 27 18:03:28,582303 127.0.0.1          127.0.0.1          RADIUS   325    Access-Request id=6
 28 18:03:28,583059 127.0.0.1          127.0.0.1          RADIUS   146    Access-Challenge id=6
 29 18:03:28,583284 5c:a6:e6:26:d9:a4  HuaweiTe_80:fc:07  TLSv1.2  64     Application Data
 30 18:03:28,592371 HuaweiTe_80:fc:07  5c:a6:e6:26:d9:a4  TLSv1.2  55     Application Data
 31 18:03:28,592937 127.0.0.1          127.0.0.1          RADIUS   307    Access-Request id=7
 32 18:03:28,593726 127.0.0.1          127.0.0.1          RADIUS   174    Access-Challenge id=7
 33 18:03:28,594017 5c:a6:e6:26:d9:a4  HuaweiTe_80:fc:07  TLSv1.2  92     Application Data
 34 18:03:28,612260 HuaweiTe_80:fc:07  5c:a6:e6:26:d9:a4  TLSv1.2  127    Application Data
 35 18:03:28,612785 127.0.0.1          127.0.0.1          RADIUS   379    Access-Request id=8
 36 18:03:28,613856 127.0.0.1          127.0.0.1          RADIUS   182    Access-Challenge id=8
 37 18:03:28,614152 5c:a6:e6:26:d9:a4  HuaweiTe_80:fc:07  TLSv1.2  100    Application Data
 38 18:03:28,631511 HuaweiTe_80:fc:07  5c:a6:e6:26:d9:a4  TLSv1.2  55     Application Data
 39 18:03:28,632063 127.0.0.1          127.0.0.1          RADIUS   307    Access-Request id=9
 40 18:03:28,632883 127.0.0.1          127.0.0.1          RADIUS   146    Access-Challenge id=9
 41 18:03:28,633165 5c:a6:e6:26:d9:a4  HuaweiTe_80:fc:07  TLSv1.2  64     Application Data
 42 18:03:28,675494 HuaweiTe_80:fc:07  5c:a6:e6:26:d9:a4  TLSv1.2  64     Application Data
 43 18:03:28,676115 127.0.0.1          127.0.0.1          RADIUS   316    Access-Request id=10
 44 18:03:28,677334 127.0.0.1          127.0.0.1          RADIUS   222    Access-Accept id=10
 45 18:03:28,677983 5c:a6:e6:26:d9:a4  HuaweiTe_80:fc:07  EAP      22     Success
```

Figure 4: phase 2 method negotiation and subsequent authentication

The first portion of the image portrays the negotiation of the inner authentication method, while the second one portrays the actual run of the authentication protocol, which is, again, EAP-MSCHAPv2. The last packet is an EAP-Success, sent by the NAS to the supplicant, signaling it that the authentication run terminated with a success and the supplicant is now granted access to the network.

4.7 Four-way handshake

This step only happens if an Access-Accept has been fired by the AS in the previous step and it is not part of the 802.1X authentication process, but it's necessary in order for the user to have access to the internet.

A **Pairwise Master Key (PMK)** is generated by the AS and sent to both the supplicant and the authenticator. The PMK, which has the same role of the shared password in WPA-Personal networks, is needed in order to perform the four-way handshake. Since a different PMK is derived in each authentication process, differently from WPA-Personal where the PMK is always the same, WPA-Enterprise guarantees one of the most important features in cryptography, which is **Perfect**

Forward Secrecy (PFS). PFS means that, even if an attacker manages to discover a PMK, it will be able to decrypt messages belonging to one and only one session. Past and future sessions are protected by other PMKs, so the attacker should recover them as well to decrypt the respective sessions.

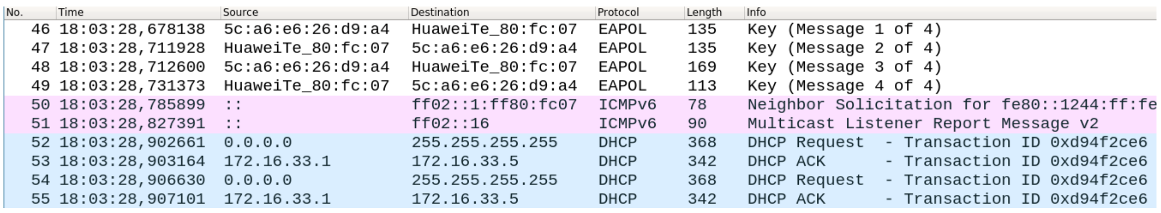
The four-way handshake is carried out outside the TLS tunnel since it only involves the authenticator and the supplicant. In order to perform this protocol, the EAPoL protocol is used once again by the two parties.

Once the four-way handshake has terminated, the supplicant is properly authenticated and possesses all the keying material to safely communicate with the access point.

Outside of the authentication, the first thing that the supplicant will ask for is an IP address, provided by a DHCP server. After receiving an IP address, the supplicant will be able to navigate on the internet freely.

4.7.1 Step 7 - example

This final screenshot contains the four packets pertaining the four-way handshake. The PMK generated by the AS is sent to the authenticator and the supplicant inside the Access-Accept packet, which was shown in the last screenshot. After this step, it can be seen that the supplicant almost immediately asks for an IPv4 address, using the DHCP protocol. An IPv4 address is instantly provided by a DHCP server, that in this case is running on the same physical machine where both the AS and NAS are running.



```
No.  Time             Source             Destination        Protocol  Length  Info
  46 18:03:28,678138  5c:a6:e6:26:d9:a4  HuaweiTe_80:fc:07  EAPOL     135     Key (Message 1 of 4)
  47 18:03:28,711928  HuaweiTe_80:fc:07  5c:a6:e6:26:d9:a4  EAPOL     135     Key (Message 2 of 4)
  48 18:03:28,712600  5c:a6:e6:26:d9:a4  HuaweiTe_80:fc:07  EAPOL     169     Key (Message 3 of 4)
  49 18:03:28,731373  HuaweiTe_80:fc:07  5c:a6:e6:26:d9:a4  EAPOL     113     Key (Message 4 of 4)
  50 18:03:28,785899  ::                 ff02::1:ff80:fc07  ICMPv6    78      Neighbor Solicitation for fe80::1244:ff:fe
  51 18:03:28,827391  ::                 ff02::16           ICMPv6    90      Multicast Listener Report Message v2
  52 18:03:28,902661  0.0.0.0            255.255.255.255    DHCP      368     DHCP Request  - Transaction ID 0xd94f2ce6
  53 18:03:28,903164  172.16.33.1        172.16.33.5        DHCP      342     DHCP ACK      - Transaction ID 0xd94f2ce6
  54 18:03:28,906630  0.0.0.0            255.255.255.255    DHCP      368     DHCP Request  - Transaction ID 0xd94f2ce6
  55 18:03:28,907101  172.16.33.1        172.16.33.5        DHCP      342     DHCP ACK      - Transaction ID 0xd94f2ce6
```

Figure 5: Four-way handshake and IP retrieval

5 Eduroam - Education Roaming

Eduroam is a project supported by campuses and universities in more than 106 territories worldwide, with more than 10,000 hotspots around the world. The purpose of eduroam is to grant access to the internet to users belonging to institutions that are part of the eduroam federation irrespectively of the campus they are visiting. A basic example is that of Erasmus students that enrolled to a university, located in their native country, but had to move to another country for study or research purposes. If both the universities they attend are part of the eduroam federation, these users will be able to use the eduroam network without needing new credentials by the hosting university; instead, they will be able to use the network with the credentials provided by their home institution. Eduroam is used without differences by students, professors and academic staff that study or work for a university taking part to the eduroam federation.

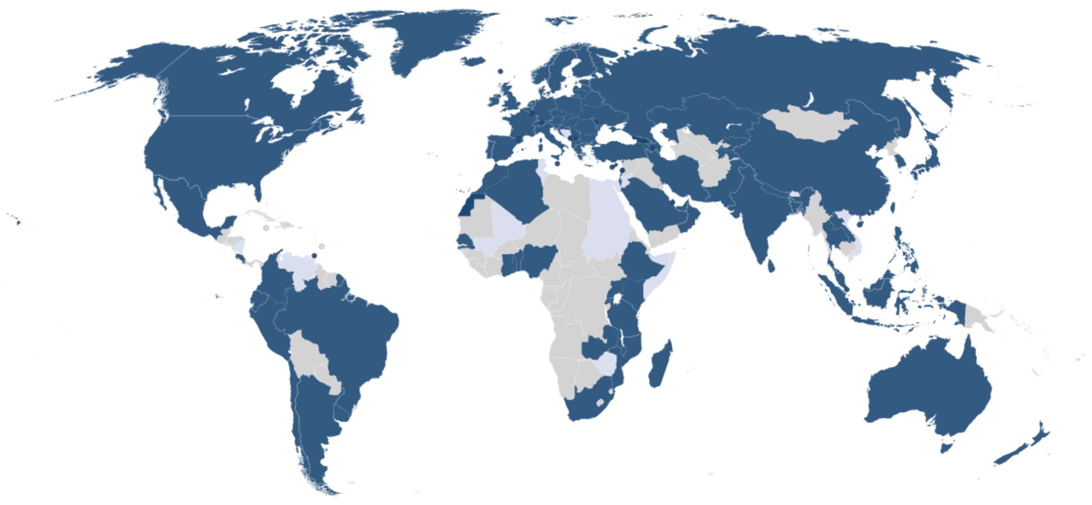


Figure 6: In blue, countries where universities take part to the eduroam federation.

Eduroam belongs to the WPA-Enterprise family of Wi-Fi networks. As said in the previous chapters, WPA-Enterprise networks are characterized by the use of user credentials such as username and password (or other identification data), in contrast with WPA-Personal networks where each user needs to know the same secret password in order to connect to the network. Moreover, as said before, WPA-Enterprise relies on 802.1X authentication, where an authentication server, an authenticator and a supplicant need to cooperate in order to grant access to the network to the user. 802.1X, if correctly used, provides a higher level of security with respect to WPA-Personal; for this reason, and for another reason which will be shown soon, eduroam is claimed to be top-notch for security aspects.

In order for a user to use the eduroam network in a foreign university, the authentication process is pretty similar to standard 802.1X wireless authentication. There is, however, an important difference that increases the security of the whole infrastructure: sensitive data, such as the username (identity) of the user and his password are never exchanged in plain text with the RADIUS server of the hosting university, meaning that this RADIUS server does not have access to these pieces of information. Whenever the RADIUS server of the foreign university receives an access request by a user that is not enrolled in that university, it will proxy the authentication request to the home server, i.e. the one of the university where the user comes from. For this reason, the authentication process sees the user's supplicant and the home server as end-points: everything in between (foreign authenticator, foreign authentication server and the internet) is just a medium used to let the two end-points communicate. This aspect of eduroam is of crucial importance, since it is responsible for letting a user connect to eduroam hotspots with the credentials of his home university and, moreover, makes it so that sensitive data can only be seen by the supplicant and the server of the user's home university.

To make the process clearer, let's suppose that there exists a user, enrolled in the University of Brescia, that has been provided with the following credentials:

- Identity: m.rossi001@unibs.it;
- Anonymous identity: anonymous@unibs.it;

- Password: password.

Let's suppose now that this student is in the United States for a research project at the Boston University. This university is part of the eduroam federation, meaning that this user can use this network with his home institution's credentials. In order to use the internet, an authentication run needs to start, where, at first, the supplicant sends the anonymous identity "anonymous@unibs.it" to the Boston University's RADIUS server. Upon reading this anonymous identity, the Boston University's RADIUS server will understand that the home university of the user is not the University of Boston, but it's, instead, the University of Brescia. To understand this, the foreign AS simply looks at what comes after the "@" symbol; this portion of the identity is also called "realm" and every institution possesses at least one. Each AS of each university that is part of the eduroam federation needs to know how to contact the RADIUS servers responsible for the various realms. At this point, the Boston University's RADIUS server will forward this request to the University of Brescia's RADIUS server, which will be told by the supplicant both the true identity and the password. Once the home RADIUS server has verified the credentials of the student, the Boston University's RADIUS server will be informed about the success, so that the supplicant will be granted access to the network.

The following picture tries to briefly sum up the example proposed above. As a disclaimer, the authentication scheme in the picture is not exactly right, but it is useful to visualize the network architecture that lets a user authenticate to his home institution from another hotspot.

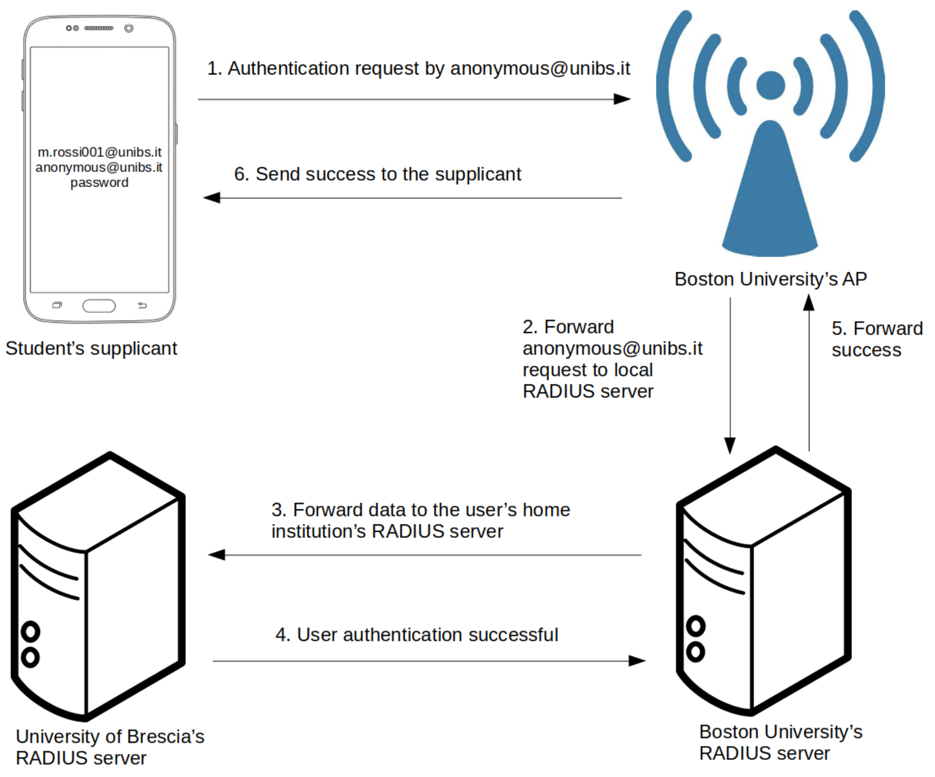

m.rossi001@unibs.it
anonymous@unibs.it
password
Student's supplicant
1. Authentication request by anonymous@unibs.it
6. Send success to the supplicant
Boston University's AP
2. Forward anonymous@unibs.it request to local RADIUS server
5. Forward success
3. Forward data to the user's home institution's RADIUS server
4. User authentication successful
University of Brescia's RADIUS server
Boston University's RADIUS server

Figure 7: Authentication of a supplicant from a remote institution.

As the developed attacks are targeting mainly mobile devices, in the following subsections it will be shown how eduroam can be configured in both Android and iOS. The configuration process on these operating systems needs to be shown in order to understand which vulnerabilities we exploited to breach users' security. We will see how eduroam, and 802.1X networks in general, can be manually configured by users, without using the eduroam Configuration Assistant Tool. This tool is really powerful as it lets universities enforce precise security policies on any device; moreover, it can be more practical for unskilled users than configuring the network manually. However, not every university has configuration profiles available on the eduroam CAT website, so not every user can use this tool. The eduroam CAT can, but not certainly does, prevent many attacks that will be shown later, so it should be strongly advertised and used by every university.

5.1 Eduroam configuration in Android

As said earlier in this document, 802.1X networks provide a really high level of security to the users, far superior to WPA2-Personal ones. However, in order for them to provide such level of security, the supplicants must be configured in the correct way. Errors in the configuration of an 802.1X network can disrupt completely the security principles offered by WPA-Enterprise.

Android, if compared to iOS, gives much more freedom to the user in terms of configuration. Moreover, the configuration of an 802.1X network on an Android device strongly depends on the Android version and on the ROM installed by the manufacturer. As we tested smartphones from different brands, we noticed that Samsung, Google, Huawei and Xiaomi devices all have different configuration windows with different rules. In general, however, when setting up eduroam on an Android phone, the user is prompted with the following fields:

- **EAP Method**. This field, which is generally set to PEAP as default, needs to be set with the EAP method indicated by the home institution. Since universities generally use tunneled methods, this field should be set to PEAP or TTLS;

- **Phase 2 authentication**. This field should contain the phase 2 authentication method needed inside the TLS tunnel, e.g. MS-CHAPv2 for PEAP or PAP for TTLS. In most Android devices, this field is left blank and marked as optional, meaning that the phase 2 authentication method advised by the AS shall be accepted by the supplicant. On newer Android versions, however, it is usually set to MS-CHAPv2 by default;

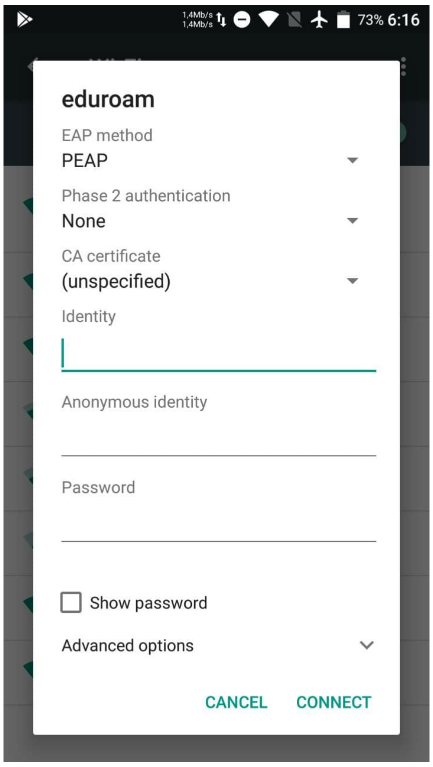
- **CA certificate**. Here, the user has the possibility to tell the supplicant to validate the AS certificate using:

  – A trusted root CA certificate (such as one from DigiCert, VeriSign, Let's

    Encrypt!, etc...). This is the default option on many smartphones running newer versions of Android;

  - A user-installed root CA certificate, for example a self-signed one provided by the university;
  - No certificate at all. This means that the supplicant will trust any certificate given to it. This is the default option for most smartphones with old Android versions and even for up-to-date Xiaomi ones;

- **Identity**. This, again, is the user's identity shown before;
- **Anonymous identity**. This field, as explained before, contains the user's anonymous identity provided by the home institution. On Android it is marked as optional since it is not strictly needed for the authentication to succeed, as the real identity can be used in its place;
- **Password**. The user's password, chosen by the user and stored in a home institution's database.


eduroam

EAP method
PEAP

Phase 2 authentication
None

CA certificate
(unspecified)

Identity

Anonymous identity

Password

Show password

Advanced options

CANCEL CONNECT

Figure 8: An example of the eduroam configuration screen of an Android smartphone

The first two fields, i.e. the phase 1 and phase 2 authentication methods, can be changed according to the home institution's directives. On older Android versions, the phase 1 method is set to EAP-PEAP by default, while the phase 2 one is set to None, meaning that any proposal by the AS will be accepted. On more recent Android versions, instead, the phase 2 method is generally MS-CHAPv2 by default.

The "CA certificate" field should always be set in such a way that the supplicant validates the AS certificate. In 802.1X networks, self-signed CA certificates are usually preferred; using a system certificate, i.e. a globally trusted CA's certificate, lets an attacker who possesses a valid certificate, signed for example by DigiCert, to set up a rogue eduroam network where supplicants will connect, since they will recognize DigiCert as a trusted CA. In other words, any certificate signed by a globally trusted CA will be automatically accepted by a supplicant. Newer versions of Android mitigate this problem by adding a new field in the configuration process, which is called "Domain"; since the attacker has a certificate which is valid for a specific DNS name (such as "*.attacker.com"), while the university's AS certificate surely contains another DNS name ("radius.domain.TLD", for example), the supplicant verifies that the provided certificate also contains the domain specified by the user, either in the commonName or subjectAlternativeName attributes of the certificate, thus rejecting the attacker's certificate since it cannot contain that domain name. Since users cannot know the value to type in this field, network administrators should always disclose the DNS name contained in the AS certificate, otherwise users won't be able to connect to the network. If a self-signed certificate is used, which, again, is the best choice for 802.1X networks as of now, the home institution should always provide the root CA's certificate to the users. Users should then download the certificate, install it through system settings and configure the supplicant to use that CA's certificate when validating the server certificate exposed by the eduroam hotspot. Some new Android smartphones have the option to specify a domain even for certificates signed by a self-signed CA, so that the smartphone will only accept one specific certificate signed by said CA. With self-signed CAs, domain isn't usually required as it is not as crucial as with globally trusted CAs, but we noticed that on some devices, e.g. Samsung's and Google's, the domain is actually required even for this kind of certificates.

Identity and password are provided to the user by the home institution. While the password can and should be autonomously changed by the user, the identity cannot.

Finally, the anonymous identity should again be set according to a university guide written by sysadmins or network managers. The anonymous identity can be any string, even a random one, followed by the right realm. Some universities, however, only accept some values as anonymous identities and not any possible string. An example of an anonymous identity for a user belonging to the University of Brescia could be *anonymous@unibs.it*. In general, this field is left blank because:

- Android marks it as optional, since it is not strictly necessary, thus making the user think it is not useful too;
- Most universities explicitly tell their users to let this field blank or to fill it in the same way as the identity field.

5.2 Eduroam configuration in iOS

As of today, iOS has a much simpler configuration window for 802.1X networks. This configuration window lets the user type just two values: identity and password. Once these fields have been filled by the user, he will be prompted with the AS certificate. Basically every field of the certificate is shown to the user, who has two choices: accept the certificate or reject it. After accepting the certificate, the user won't be prompted with the certificate until it deliberately chooses to forget the network or the AS certificate changes, for example because it expired and needed to be signed again. As it can be understood, it is up to the user to verify whether an X.509 certificate is genuine or not.

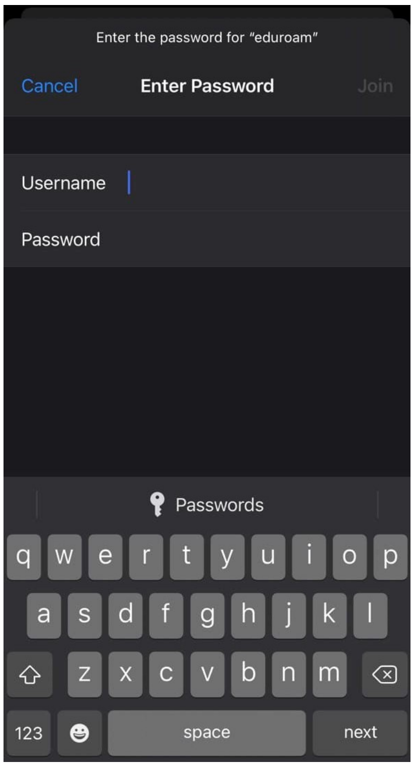


Figure 9: The configuration window for eduroam in iOS.

Eduroam configuration in iOS is much simpler than in Android and certainly more user friendly. However, the lack of freedom makes some attacks easier to mount, since the user has no control on what the device does.

6 Vulnerabilities tied to eduroam in mobile OSs

Now that every aspect of 802.1X authentication and eduroam configuration has been covered, the major flaws in eduroam's security can be thoroughly discussed. As it will be shown in the following subsections, most of them are related to a wrong configuration of the network, be it for a user's fault (Android) or for OS constraints which don't allow enough freedom during the configuration (iOS). The attacks that will be shown in the following paragraphs can be done against any 802.1X network, so they are not related to aspects unique to eduroam.

6.1 Lack of anonymous identity

The purpose of the anonymous identity is to guarantee to users that their true identity will be disclosed to a trusted authentication server only, i.e. their home institution AS in eduroam. As a matter of fact, once a supplicant receives beacons by an access point advertising a known 802.1X network, it will for sure exchange at least its anonymous identity, as it was shown in section 4.1. If an anonymous identity is not set, the real user identity will be used in its place, thus compromising user's privacy as he can easily be tracked in space and time.

Anonymous identities should always be used to guarantee a certain level of privacy to the users. However, most universities' guides tell the users to leave this field blank. Since 802.1X works fine without anonymous identities, network administrators too tend to close an eye on this aspect.

As said in section 5.1, Android lets the user fill the anonymous identity field. Again, this field is marked as optional, making the user think it is not useful, and universities tell users to leave it blank intentionally. On the other hand, iOS doesn't let the user use an anonymous identity without a configuration profile. For this reason, iOS devices will almost always disclose the real identity to any eduroam access point, even rogue ones.

6.2 No server certificate validation

Among all of the vulnerabilities that will be discussed, this surely is the most dangerous one, since many attacks can be carried out if this condition is met.

As mentioned before, in Android the user has the possibility to configure the supplicant so that the server certificate is not validated at all. In this way, any digital certificate exposed by an authentication server will be trusted by the supplicant, thus making it establish a connection even with a non-trusted party.

As imposed by Google in December 2020, the option to not validate the server certificate should not be present in Android starting from version 11. As of today, even with Android 12, most of the Android smartphones still have this option available, exception made for Google's Pixels. What makes this problem even bigger is that a great amount of universities, including ours, instruct the user to use the "Do not validate" option.

6.3 Manual validation of the server certificate

This kind of vulnerability only affects iOS devices, as they are the only ones asking the user to approve or reject a certificate.

As said briefly before, the first time a user with an iOS device connects to eduroam (or an 802.1X network in general), after confirming its credentials he will be prompted with the server certificate. Here, the user can read most of the fields contained in the X.509 certificate, such as the common name (CN), the state, region, city and other parameters. What is expected from the user is a manual verification of the various fields; if the user deems the certificate to be true, he should accept it, otherwise he should reject it. The main problem here is that X.509 are easy to falsify by a malicious user. For instance, every field of a real certificate can be copied and used in another self-signed certificate, with the only main exceptions being the public key (which can be falsified, but it makes no sense to do so since the attacker does not possess the corresponding private key) and, of course, the signature of the certificate. So, in order for a user to discriminate between a fake and a real certificate, he should be able to remember by heart the public key of the original server certificate, which is impossible.

In iOS, just like in Android, the user has the possibility to manually install a root CA certificate on his device. However, even if the right root CA certificate is installed, there's no way to tell the device to use that certificate for an 802.1X network. In the end, even if the user installs the root CA certificate, the server certificate will still be prompted to him, awaiting for manual approval.

As it can be easily understood, a user prompted with a new certificate by an eduroam hotspot can easily be tempted to accept it. The user may think that the device forgot the network, or that the university's network admins had to make another certificate, for example because the previous one expired. A little bit of social engineering and user inexperience play a big role in this context, but if iOS allowed users to use a downloaded root CA certificate (or a globally trusted one) for automatic validation it would be much more secure.

In sections 13 and 14 the genuine certificates are compared with the falsified ones. As it can be seen, they are virtually identical, aside from some little details that a human can hardly detect. Of course, the higher the similarity, the greater the confidence of the user to accept the fake certificate as the real one.

6.4 Insecure implementation of MS-CHAPv2 in iOS

As already said before in this work, among the authentication protocols proposed here, MS-CHAPv2 is one of the few that provides mutual authentication. This means that the supplicant first proves to the AS that it knows the password; after that, it is the turn of the AS to show the supplicant that it knows the password too. If the AS answers with a wrong response to the challenge sent by the supplicant, or doesn't answer at all, the supplicant should refuse to go any further, as it means that the AS doesn't know the right password. Any supplicant we tested works as described here: if we try to force an Access-Accept during an MS-CHAPv2 authentication, it refuses it. However, we discovered that iOS devices, even with the most recent iOS version, do not work this way: an Access-Accept packet will be accepted even halfway through the MS-CHAPv2 run. This behaviour is a serious security threat as a rogue authentication server can correctly authenticate iOS devices without knowing the password and even steal sensitive, password-related data.

6.5 No phase 2 method configured

As it was shown in 4.5, after phase 1 authentication the server proposes an authentication method to the supplicant. If the supplicant is not set with a specific phase 2 method, then it will accept what the AS proposed in the previous message. This can pose a big threat to the users, as an attacker can set a weak method (e.g. EAP-GTC) as default on his AS. Whenever the user decides to connect to the network, if it proceeds until phase 2 authentication, it will be easy for the attacker to obtain the password without any effort. On the other hand, setting the supplicant to use a specific, stronger phase 2 authentication methods makes password stealing harder or even infeasible.

Both Android and iOS devices can present this vulnerability. Older Android devices, as shown in figure 8, have the phase 2 field set to "None" by default. More recent versions of Android, however, have this field set by default to "MS-CHAPv2", which is the strongest among the most commonly supported EAP methods; unluckily, albeit being one of the strongest, MS-CHAPv2 has been proven to be weak against a variety of attacks. iOS devices, on the other hand, have both phase 1 and phase 2 authentication methods set to ”None” and they cannot be modified by the user. This means that, without using external tools provided by Apple or eduroam CAT, every iOS device will authenticate according to the methods proposed by the AS.

7 Attacks studied and tested

In this section, some possible attacks against the users of eduroam will be presented and discussed. Beware that, although the network under attack was eduroam, all of these attacks can be generalized to most of 802.1X network as they are not strictly tied to eduroam, but rather to the 802.1X authentication process and to the configuration of such networks on mobile devices.

The attacks shown here are just some of the possible ones that we intended to investigate on. These attacks are the ones we developed and tested in a controlled environment, just to understand under which conditions a user could fall into a specific trap. Despite having permissions granted by the ICT team of our university, the attacks we decided to implement for a live testing with unaware users were purposefully weakened, in order to demonstrate the feasibility of such attacks without actually stealing sensitive data. Every detail regarding the live experiment that we did will be thoroughly shown in section 8.

7.1 User tracking

As said before, the anonymous identity is used to prevent users from disclosing their real identity to authentication servers different from their home institution's one. However, as already mentioned before, the use of an anonymous identity is generally not endorsed by universities. This results in users disclosing their personal identity with whatever access point advertising an eduroam network, since the inner identity is sent in the first steps of the authentication process if no anonymous identity is set. As already shown before, in eduroam networks the identity of a user usually (but not always) has the form of ⟨name.surname@university.TLD⟩. This means that the 802.1X identity usually contains enough pieces of information about the user to let an attacker identify the user in the real world. If an attacker manages to set up an access point advertising an eduroam network, nearby supplicants that have their Wi-Fi on will try to connect to it. If the users did not set their anonymous identity, the attacker has for sure access to the real identities of the users, thus giving him knowledge of time and place where a user was located at.

Both Android and iOS are susceptible to this attack, with one main difference: while iOS devices, as of now, cannot defend themselves against this attack since there is no way to set the anonymous identity unless by means of eduroam CAT, Android devices can. For Android devices to defend themselves, however, universities would need to start endorsing the use of anonymous identities to their users. Since anonymous identities are rarely advised by universities, Android devices are as likely as iOS ones to fall in this attack.

7.2 Credential stealing

This attack has the goal to steal sensitive data from the users, such as usernames and passwords. In most universities, these credentials are the same ones used for accessing the online services provided by the university. In the case of the University of Brescia, with these credentials a user can access its esse3 profile, an online service where the student can pay taxes, subscribe to exams, look at their results, access his e-mail account and so on and so forth. The same goes for professors, who have access to many more institutional services. Moreover, since users tend not to diversify their passwords among different services, it may happen that a user has his eduroam password used for other services, like social media, money transfer applications, etc... This is of course a major threat, as the attacker could then try, and maybe succeed, to log in various services with malicious purposes.

This kind of attack can be carried out in a completely passive way (without user interaction) and in an active way (with user interaction). If the attack is passive, then only Android devices can fall into this kind of trap and only under these conditions:

- The device's Wi-Fi is on;
- Automatic access to known networks is on;
- The user set his smart device with the "Do not validate" option.

Under these conditions, the user's device will go past the phase 1 authentication step, since it doesn't validate the certificate provided by the evil-twin, and will proceed with phase 2 authentication.

Tunneled methods, i.e. methods which occur inside of a TLS tunnel, are known to be problematic if the tunnel was set up with a non-trusted party. In more detail:

- PAP gives out the password in plain text, thus allowing the attacker to simply catch it and store it;
- EAP-GTC, which is strongly based on PAP, shares with it the fact that the password is sent out in the clear;
- MS-CHAPv1, although it sends a quantity which is password (and user) related, but not plain text, can be easily broken and many attacks to retrieve the password were developed against it;
- MS-CHAPv2, which is a stronger version of the previous one, can be easily broken too and breaking it has the same complexity as breaking a 56 bit DES key, which is far too easy today. Among these methods, MS-CHAPv2 surely is the most annoying one to deal with for an attacker, but this doesn't mean that the attacker cannot deal with it.

As a result, if a user's Android device is misconfigured and is using one of these phase 2 authentication method, then the user's password can be recovered by the attacker with variable difficulty, depending on the phase 2 method set by the user.

As it was shown previously, 802.1X authentication requires that the authentication server proposes both a phase 1 and phase 2 authentication methods, which can be agreed or rejected by the supplicant. Since most of the Android devices, at least the older ones, have the phase 2 authentication field which is set to "None" by default, it means that they will accept the phase 2 method proposed by the AS. For this reason, in order to make the attack easier on the attacker side, the AS running on the evil-twin can be set to propose EAP-GTC for phase 2. Proposing EAP-GTC for phase 2 is the most interesting choice as it is supported both in EAP-TTLS and in EAP-PEAP and, most importantly, gives the attacker the password in plain text. If the supplicant has the phase 2 authentication method set to "None", the supplicant will agree on using EAP-GTC, thus gifting the password to the attacker. On the other hand, if the supplicant is set to use a specific phase 2 authentication method (different from EAP-GTC), the AS cannot enforce EAP-GTC but, as said previously, practically every other tunneled method is broken too so the password can still be recovered.

Active attacks require for the user to interact with his device, unlike passive ones where a user can have his smartphone locked in his pocket. In active attacks, the user intentionally tries to connect to the rogue network, even if the smartphone did not connect to it automatically. On a correctly configured Android phone, once the AS provides its certificate to the supplicant, the supplicant will try to validate it and fail, since the certificate provided by the AS was not signed by the same CA whose certificate is installed on the smartphone. Some Android smartphones, however, show this behaviour: other than showing the error message, they also give the option to the user to "Connect anyway". If the user presses this button, the option to validate the server certificate will be automatically downgraded to "Do not validate". This results in the user being unable to connect at first, but succeeding the second time, thus falling into the attack. iOS devices are only vulnerable to this kind of attack, as if they sense a change in the server certificate accepted the first time by the user, they don't connect to the network and prompt the new certificate on screen to the user, waiting for approval. However, since users cannot easily distinguish between a fake and a real certificate as they can be way too similar, they might be tempted to accept the new certificate and connect to the network. Moreover, since iOS devices always do the phase 1 and phase 2 authentication methods advertised by the AS, setting the AS with EAP-GTC as default method for phase 2 results in iOS users being forced to disclose their passwords in plain text. Active attacks are harder to carry out as they require for the user to be using his smartphone and actively looking for Wi-Fi networks to connect to. However they shall still be considered as a threat.

7.3 User authentication without a priori knowledge

The authentication of a user on the evil-twin cannot be considered an attack per se, but rather the starting point for a series of other attacks. If user authentication succeeds, then the user will start using the rogue network, ultimately paying the consequences.

Historically, attacks to 802.1X networks only aimed at stealing users' sensitive data, using the techniques explained in the previous attack. However, in order to pose other threats to the users, we wanted to go further and see under which conditions a user could be successfully authenticated. As a matter of fact, the only big difference between the real eduroam hotspot and an evil-twin, besides the certificates, is that the real eduroam hotspot has access to a database containing the passwords of the users. When authentication starts, the AS can read those passwords and check if the ones provided by the users correspond to the ones in the database. As attackers, we don't have this possibility, as we don't know the passwords in advance, unless we previously recovered them with a credential stealing attack. In some cases, however, it is possible to both steal the password and guarantee access to the network in the same authentication run, while in others it is only possible to do one of the two things.

In order to carry out this kind of attack, what matters is firing a forced Access-

Accept packet from the RADIUS server at the right time during EAP authentication. If a supplicant receives an Access-Accept at the right time, it will think the authentication was successful and it will connect to the network.

As already said previously, phase 2 authentication methods can provide either one-way or mutual authentication. As a matter of fact, PAP, EAP-GTC and MS-CHAPv1 all are one-way authentication methods. This means that the supplicant computes some quantity and sends it to the authentication server, then it waits for a boolean response: accept or reject. These methods have shown to be particularly weak as they let us both obtain the password and authenticate the user, thus maximizing the damage. On the other hand, among the methods discussed previously, MS-CHAPv2 provides mutual authentication. In MS-CHAPv2, the supplicant receives a challenge from the server, computes some quantity and sends it back to the AS along with another challenge. Now, the supplicant does not wait for a boolean response only, but it expects for the AS to compute the correct response based on the password, the username and the challenge sent by the supplicant. For this reason, the supplicant will reject an Access-Accept fired in the middle of an MS-CHAPv2 run. This makes it impossible for us to both steal the password and authenticate the user at the same time, unless the device runs iOS since, as it was shown in 6.4, iOS devices do not validate the MS-CHAPv2 server response, thus letting the attacker steal both sensitive data and force an Access-Accept. However, access to the network can still be granted to the user: whenever the rogue AS sees that the supplicant wants to do MS-CHAPv2, it can fire an Access-Accept before starting the authentication protocol. In this case, the supplicant will accept it and the phase 2 authentication will be completely bypassed. In summary, if a supplicant wants to do MS-CHAPv2, the attacker can choose between catching sensitive data or authenticating the user, but cannot do the two things in the same authentication run.

The outer authentication method should not be a concern. However, we found out that FreeRADIUS behaves differently when using EAP-PEAP and EAP-TTLS. In the first case there is a blatant negotiation stage, so that if MS-CHAPv2 is proposed by the supplicant, the attacker can discover it before starting the authentication run and fire the Access-Accept, thus resulting in a successful authentication. In EAP-TTLS, instead, it seems that the supplicant starts the authentication run as soon as it enters the TLS tunnel, thus giving no option to the AS but to proceed with the authentication. This, of course, results in an error as the rogue AS does not know the user's password and cannot compute the cryptographic material needed for server-side authentication. We weren't able to bypass MS-CHAPv2 inside EAP-TTLS, but this does not mean that it cannot be done. We know for sure that FreeRADIUS can be tweaked to bypass phase 2 authentication in this case too, since the AS must send a challenge first so it must be notified about which protocol needs to be run.

In summary, the following table shows which combinations of methods, according to our tests and experiments, let the attacker steal the password (or quantities related to it which can be broken), authenticate the user or both.

Methods combination	**Steal password?**	**Authenticate?**	**Both in a run?**
PEAP + MS-CHAPv2	Yes	Yes	No (yes in iOS)
PEAP + GTC	Yes	Yes	Yes
PEAP + PWD	No	Yes	No
TTLS + PAP	Yes	Yes	Yes
TTLS + GTC	Yes	Yes	Yes
TTLS + MS-CHAPv1	Yes	Yes	Yes
TTLS + MS-CHAPv2	Yes	No	No (yes in iOS)

Table 1: Combinations of phase 1 and phase 2 authentication methods we tested and if we could steal the password (or data related to it), authenticate the user or to do both in the same authentication run.

7.4 Device de-anonymization

An attacker could be interested in having more pieces of information about the supplicant itself, i.e. its hardware and software. A deeper knowledge of the supplicant not only makes tracking attacks shown previously more reliable, but could potentially make the attacker spoil vulnerabilities tied to the hardware or the OS version the supplicant is running on. In order to do so, we discovered that this kind of information can be potentially obtained by analysing:

- MAC address. This is doable even if the supplicant does not authenticate on the evil-twin because it was correctly configured. However, the evil-twin will know the MAC address of the supplicant because it can be caught during the initial phases. If a randomized MAC address is used, then the attacker cannot go back to the vendor of that device, otherwise it is totally possible and easy to do so;

- DHCP lease. If the supplicant connected to the network, it surely asked for an IPv4 address, which was then provided by the DHCP server running on the evil-twin. Inside the IP leases file, some pieces of information provided by the DHCP client on the supplicant can be found. These data can contain, for example, the OS version of the supplicant and the hostname advertised by the supplicant itself. The hostname field usually gives the attacker information about the model of the smartphone and, possibly, of the user, as common hostnames have the form of "⟨smartphone model⟩ of ⟨user's name⟩";

- DNS queries. As soon as the supplicant connects to the rogue network, it will start doing DNS queries to pre-defined servers. iOS devices start querying Apple servers, Huawei devices do the same with Huawei servers (HiCloud), and so on and so forth. By analysing the DNS queries, the attacker can go back to the vendor of the device.

It can be easily seen that while a user does not connect to the evil-twin, as the supplicant was correctly configured, the attacker cannot obtain data about that user's device for sure and if it can, it can only have knowledge of the vendor of the device. If it authenticates to the evil-twin, instead, the attacker can usually determine vendor, model and OS version of the smartphone used.

7.5 Traffic sniffing

Once the user is connected to the rogue network, its device will start exchanging traffic with the rest of the internet. As the device is connected to the rogue access point, all the traffic, be it exchanged with local devices or remote ones, passes through the evil-twin. Packets can be easily logged by the attacker and analyzed later. The most interesting data the attacker can collect surely are:

- DNS queries. The attacker has complete knowledge of which websites the user under attack is visiting. Each time a user opens a link found on a search engine, such as Google, the device will try to resolve the symbolic name of the website into its IPv4 address by making a DNS query to a DNS server (which usually is the access point). Moreover, installed apps such as Whatsapp, Instagram, Facebook and TikTok will try to contact their respective servers, thus giving the attacker knowledge of which applications a user has installed;
- HTTP traffic. Nowadays, most of web servers use HTTPS, meaning that any data exchanged between a client and a server is end-to-end encrypted using TLS. However, there are still websites that use plain HTTP, meaning that data is exchanged in the clear between client and server and can be easily sniffed by the evil-twin. This includes not only data contained in web pages, which may not be interesting for the attackers, but also login credentials used by a user to authenticate on a website.

Moreover, we noticed an interesting thing: while most of the smartphones' browsers contact servers using HTTPS (TCP port 433) by default, some of them use HTTP instead (TCP port 80). This means that once a supplicant obtains the IPv4 of a web server, it will try to connect to it via HTTP; if the web server supports both HTTP and HTTPS, it will answer using HTTP as it is the protocol chosen by the client. In this case, even if a website supports HTTPS as well as HTTP, the security of the user will be compromised because of the policy of his browser.

To summarize, if a user connects to the evil-twin, many pieces of information can be gathered by the attacker, such as:

- Websites visited;
- Applications installed;
- Data sent and received through HTTP.

7.6 DNS spoofing and phishing attacks

DNS spoofing is the act of intentionally translating symbolic names (such as www.google.com) to IP addresses chosen by the attacker instead of the right ones. Whenever a user wants to visit a website whose name's translation is manipulated by the attacker, there are two possible outcomes:

- The client queries the obtained IP address using HTTP. In this case, the user will simply see the wrong website without errors;
- The client contacts the web server via HTTPS. In this case, things become harder for the attacker, since the client expects a valid TLS certificate containing the same symbolic name that the client wanted to resolve. A globally trusted TLS certificate respecting this condition cannot be possessed by an attacker, as globally trusted certification authorities won't release one to the attacker

without proof that he is the owner of that particular domain. On the other hand, the attacker could produce a self-signed certificate containing that domain name; this certificate won't be accepted by the client, as it is not signed by a globally trusted CA, but the user can usually proceed anyways to its own risks.

During our tests, we discovered three categories of mobile OS browsers: ones that use HTTP instead of HTTPS by default, ones who use HTTPS but let the user proceed on a website even if the TLS certificate cannot be trusted and ones who use HTTPS and forbid the user from stepping further, thus blocking him from falling into the trap.

As it was made clear, nowadays DNS spoofing is pretty hard to carry out because of the wide adoption of HTTPS in place of HTTP. This is a good thing, of course, since users can browse the internet more safely. However, browser's policies and psychological factors can still pose a threat to the users.

Phishing attacks can come strictly paired with DNS spoofing attacks. The attacker could redirect the user to a fake website, resembling the true one, but that is under control of the attacker. Luring a user on a fake website can make him disclose secrets, for example passwords, relying on the user inexperience and inattentiveness. Phishing attacks heavily rely on the ability of the attacker to convince the user to give him sensitive data, so the technical aspect has little-to-none importance in this context, while the psychological aspect is, once again, predominant. As shown before, the outcome of these attacks depends on the web browser too. If a browser uses HTTP by default, then the user is strongly at risk, as the fake website will be presented without warnings. On the other hand, a web browser that uses HTTPS but lets the user open the website at his own risks shifts the responsibility on the user. Finally, browsers that don't let the user navigate on a website with a self-signed, non-globally trusted certificate, surely make phishing attacks way too hard if not impossible to be implemented.

8 Live experiment

After testing some attacks in a controlled environment, we wanted to do a live test on the field. The purpose of this test was to effectively understand how many real-world users were prone to falling in these traps. In the following subsection, we will discuss the main goals of our live experiment, along with the hardware and software used to carry it out. When talking about the software, some considerations about their configurations will be made so that it will be easier to understand how these programs can be tweaked to enable some attacks. We think it is important to show how to use these tools to perform attacks, as literature is strongly lacking of proper, deep explanations.

After consulting us with the ICT team of our university, we decided to put under attack another network instead of eduroam. This network is called "Studenti" (italian for "students") and is the Wi-Fi network that can be used by students, who can login with their credentials provided by our university. The main reasons behind this choice are:

- We did not want to frighten foreign students who would have then contacted their respective universities for clarifications, which did not know anything about this experiment;
- There are very few foreign students in our university, so the users who use eduroam are mainly users from our same university;
- Erasmus students are provided with credentials by our university, so they might end up using Studenti too instead of eduroam;
- The students from our university generally use the network Studenti rather than eduroam. The ICT team told us, by looking at real time data, that the number of users using Studenti is ten times higher than the one of users using

"eduroam". This means that we could have caught much more data attacking the former instead of the latter.

To summarize, the users under attack would have been virtually the same in both cases (Studenti or eduroam) with the main difference that there are far more students in our university who use Studenti instead of eduroam, thus letting us catch more data.

8.1 Goals

The main goal of our experiment was to see how many devices would fall into some of the attacks discussed in the previous section. Catching data from real-world users would give us a measure of effectiveness of the attacks we developed. Since we supposed that a huge number would fall at least in some of the attacks we prepared, a live experiment was the perfect tool to verify our hypothesis.

In detail, we wanted to investigate mainly two things:

1. How many users don't use an anonymous identity, thus exposing themselves to tracking attacks;

2. How many users would connect to the rogue network, be it for a bad configuration of their device or for explicit choice. Among these users, it was also of our interest to analyze which phase 2 authentication method was set on their devices, letting us know how easy for an attacker would be to steal their passwords.

As it was shown in section 7, while collecting access attempts from users is a fairly easy task, authenticating them is harder as it requires deeper knowledge and tweaking of the programs used by the evil-twin. However, authenticating the users was our biggest goal and so the part of the project where we concentrated the most. In

Appendix C (section 15) it can be seen how we modified FreeRADIUS to successfully authenticate any user.

In order to grant anonimity to every user, we decided to hash the username in both the anonymous and personal identity, but not the realm (if there was one). In order to check whether a user was using an anonymous identity or not, we simply checked that the hashed anonymous identity was different from the hash of "anonymous" (or some variations of it), which should be the value used worldwide for anonymous identities. The hash function used to hash usernames is SHA256, which is, as of today, a strongly secure hash function.

In order to carry out this live test, we decided to set up a number of different evil-twins, all of them configured in the same way, and to place them around our university to collect data. In the next paragraphs, the actual physical devices we used as evil-twins along with the software needed to achieve our goals will be shown.

8.2 Hardware

In order to make the attacks easier to carry out and more appealing for an attacker, hardware should be inexpensive, small in volume, lightweight and with little power consumption. These features are wanted by the attacker in order to make the attacks physically feasible. In order for the attacker to trap users on a vast area, more access points are needed to cover it completely, so hardware should be inexpensive. Moreover, small and lightweight hardware can be easily hidden in some places to be retrieved later for data analysis. Since it can be assumed that the evil-twins are left out in the wild or used outside of the attacker's house, they need batteries or power banks to work; for this reason, they should be less energy-intensive as possible, in order not to burn out the energy supply too soon.

For this reason, the hardware used in our attacks consists of mainly two items:

- Raspberry Pis, in our case the 3 model B+;

- TP-Link Archer T2U network adapters.

Raspberry Pis provide enough computational capabilities to run the software needed by the evil-twin to recreate the 802.1X network to attack. Moreover, as of today, the advertised price for a recent model of Raspberry Pi sits around 40-50 euros, thus making them pretty inexpensive. Raspberry Pis run Rasperry Pi OS (named Raspbian OS before 2020), which is a Unix-like OS based on Debian, meaning that it can be freely configured and there's plenty of software for it to be installed. Raspberry Pis are pretty small and lightweight. With a case on, a Raspberry Pi 3 model B+ measures around 9x7x3 centimeters, weighing about 105 grams. This means that it can be easily hidden without it being noticed. As for the energy consumption, under heavy workload a Raspberry Pi 3 model B+ should require around 5 watts of power. This means that using a power bank with a capacity of 10000 (ten thousand) mAh, a fully stressed Raspberry Pi 3 model B+ can be used for around ten hours. For our tests, energy consumption was not a problem as we had the opportunity to plug the Raspberry Pis to the power outlets of our university. As an attacker may not have access to the university or to a power outlet in general, energy consumption becomes a bigger concern as he has to rely on power banks, which have a limited power supply.

Since the aim of the attacks was to replicate an 802.1X hotspot, it is clear that hardware capable of performing 802.1X authentication was needed. Unfortunately, the Rasberry Pi integrated wireless interface cannot handle such kind of authentication on the server side as of now. For this reason, a USB wireless interface was needed. Since it was assumed that this behaviour of the integrated wireless interface was linked to its drivers (Broadcom), the TP-Link Archer T2U family of adapters was an effective choice, since its drivers are provided by Realtek and they support 802.1X authentication. Even if these network adapters proved to be suited for this work, the drivers provided by Realtek are not Linux compatible, so we had to resort to third party drivers found on the internet. Resorting to other antennas, such as ones powered by Atheros drivers, would have probably been an even better choice. Again, the antennas we used cost between 10 and 15 euros on major e-commerce platforms, so they are pretty inexpensive too.

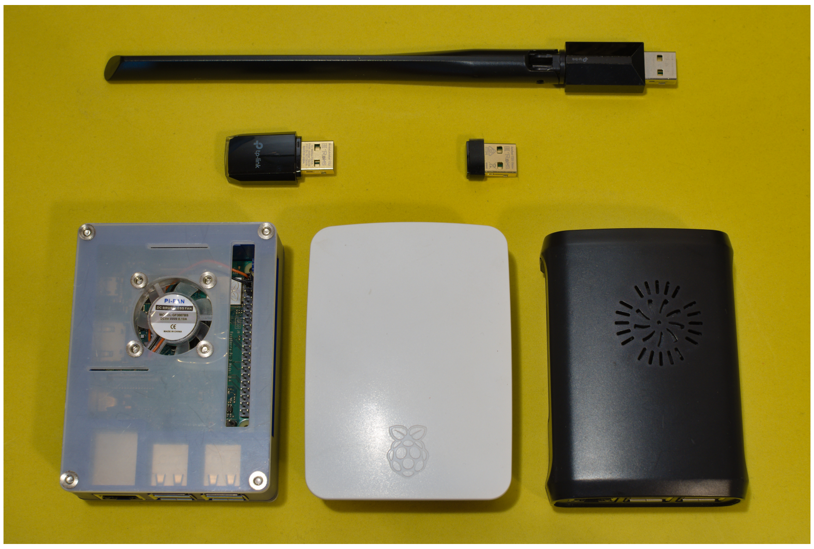


Figure 10: The three Raspberry Pi 3 model B+ along with the three different TP-Link Archer T2U dongles we used in our live test.

8.3 Software

Besides hardware, which is surely important for the feasibility of the attacks, software plays a huge role too. To build a complete evil-twin, we had to work with a variety of different software, each one of them playing an important role in the whole scheme. All of the software presented here are free and open source, an important aspect since an attacker has only to deal with hardware costs if he wants to set up one or more evil-twins.

8.3.1 FreeRADIUS

The most important actor in 802.1X authentication surely is the authentication server, usually known as RADIUS server since it generally uses the RADIUS protocol to carry out authentications.

FreeRADIUS is a really famous software that can be used for this purpose and is the one used all over the world for real eduroam hotspots.

In order to be fully functional and to make EAP-TTLS and EAP-PEAP usable for 802.1X authentication, FreeRADIUS needs at least one X.509 certificate, i.e. the server certificate. Since our university uses a self-signed X.509 certificate, we had to generate two different certificates: one for the certification authority and one belonging to the authentication server. FreeRADIUS comes with some useful scripts that make automatic certificates' generation really easy, so we used them to make our own certificates. The certificates generated by these scripts can be customized by editing the corresponding configuration files. Our goal was to make our certificates look as similar as the real ones, so that any user seeing the certificates would think they were the genuine ones. In order to do so, we first took a look at the real certificates provided by our university, then we edited the configuration files accordingly. In section 13 the certificate of the true root CA of our university is shown side by side with the one we did generate. As it can be seen, other than some details, the two are practically identical for a user. In section 14, instead, there is a side by side comparison of the two server certificates, which are the ones actually presented to the smartphone by the authentication servers. As it can be noted, these two also are virtually identical.

The first modification we did to FreeRADIUS was intended to log every access attempt happening outside the TLS tunnel, in the first phase of the 802.1X authentication. Here, data such as current time, MAC address of the supplicant and the anonymous identity were written to a log file to be analyzed later. This modifications were intended to let us analyze whether a user was using an anonymous identity or not.

Since our major goal was to authenticate as many users as we could on the evil-twin, deeper modifications to FreeRADIUS were needed. Whenever a supplicant passes phase 1 authentication, be it for wrong configuration of the device of for user error, a TLS tunnel is built to encrypt data end-to-end between supplicant and AS. To authenticate the users, we tweaked the portion of code that FreeRADIUS runs inside the TLS tunnel in such a way that an Access-Accept packet is fired at the right time. As mentioned before, Access-Accept packets fired at a wrong time make the supplicant drop the connection, resulting in a failed authentication attempt. In order to maximize the odds of correctly authenticating a user, we decided to propose EAP-PEAP for phase 1 and EAP-GTC for phase 2 authentication. More details about these choices can be found in sections 7.3 and 15. According to our modifications, FreeRADIUS was set to:

- Fire an Access-Accept as soon as it could, if the supplicant was set to use EAP-TTLS for phase 1;
- Propose EAP-GTC for phase 2, if the supplicant agreed on using EAP-PEAP. Then:
  - Do a legit run of EAP-GTC and then fire an Access-Accept, if the supplicant agreed on doing EAP-GTC;
  - Wait for an EAP-NAK+Desired-method and then force an Access-Accept, if the supplicant did not agree on using EAP-GTC.

Basically, inside EAP-PEAP we were able to authenticate users with every kind of phase 2 method, since we could either break EAP-GTC (in case the supplicant agreed on using it) or bypass phase 2 (in case the supplicant wanted to use another EAP method). Inside EAP-TTLS, things are quite different as it seems that we don't have time to negotiate the phase 2 method, so we forced an Access-Accept as soon as we entered the tunnel, hoping for the supplicant to accept it. It must be noted that in EAP-TTLS, only users who set MS-CHAPv2 as phase 2 method couldn't be authenticated, as all the other common methods inside EAP-TTLS (PAP, MS-CHAPv1 and EAP-GTC) provide one-way authentication, thus will accept a forced Access-Accept packet. Beware that in EAP-PEAP, when the supplicant first sends its identity inside the tunnel, the AS can fire an Access-Accept as soon as it receives the EAP identity. Doing this ensures that every phase 2 method inside EAP-PEAP leads to a successful authentication run, but has a major flaw for our purposes: the attacker cannot know which phase 2 authentication method was chosen by the user, as phase 2 is completely bypassed before even negotiating a phase 2 authentication protocol. One of our goals was to discover which authentication methods were used by users, so firing an Access-Accept as soon as the inner identity was received was not a good idea. We preferred, instead, to propose to the supplicant a method we could break (EAP-GTC) so that we could either break it or know which method was chosen by the supplicant and, only then, bypass it.

8.3.2 HostAP daemon

In order to make the Raspberry Pis advertise an 802.1X network with SSID "Studenti", we installed and configured hostapd, a widely known software in the GNU/Linux community. No special modification has been made to this software, as we were able to configure it according to our needs by editing the configuration files. Here we specified the address and port of the RADIUS server, which would have been contacted by hostapd to let supplicants perform 802.1X authentications.

Moreover, in order to forbid users from trying to authenticate too many times, some scripts were written to populate a MAC address blacklist used by hostapd. Whenever a supplicant reached the maximum number of authentication runs permitted, its MAC address was inserted into the blacklist and dynamically reloaded through hostapd_cli, a hostapd command-line tool that lets other program interact with hostapd.

8.3.3 ISC-DHCP-Server

In order to make the evil-twin look more like the a genuine hotspot, we needed to provide our Raspberry Pis with a DHCP server. This program would be able to provide supplicants with IPv4 addresses after authenticating, thus letting them use the internet. As it was shown in subsection 4.7, as soon as the Four-Way Handshake completes, the supplicant will ask the access point for an IPv4. If an IPv4 is not provided by the access point, smart devices tend to disconnect from the network as they cannot use the internet.

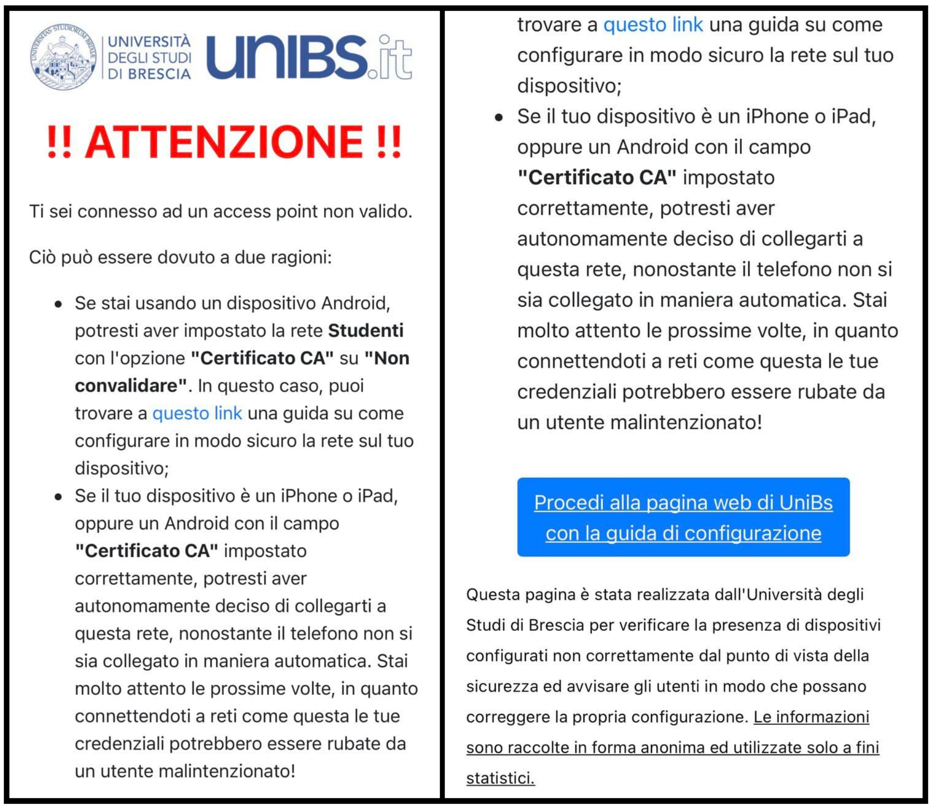
To achieve this goal, we installed the ISC-DHCP-Server software, which provides a complete DHCP server. Then we defined an IPv4 address pool to take IPv4 addresses from. In our case, this subnet was 172.16.33.0/24. The external network interface of each Raspberry Pi was configured to use 172.16.33.1 as its own address and ISC-DHCP-Server was set to listen for DHCP requests on the USB wireless interface.

8.3.4 Captive portal

Upon correctly authenticating to the rogue eduroam hostpot, we wanted to let the users know that either their smartphones were badly configured or that they decided to access an evil network. In order to do so, we decided to set up an informative captive portal, i.e. a web page that forcefully opens after the user connects to a certain network. Captive portals are widely used for hotspots in public places but practically never used in 802.1X networks as authentication is performed at a lower level.

In order to do so, we developed a small website in Python, using the micro-framework Flask. This website would present the user a web page containing a warning, plus some tips on how to configure the connection to the network to improve the overall security. The web page is completely in italian, as the users under attack belonged to our university.

To show the user this captive portal, we used dnsmasq to redirect specific DNS query to the local hosted web server. At this point, whenever a user connected to the network, he was prompted with the warning web page.


UNIVERSITÀ DEGLI STUDI DI BRESCIA unibs.it

!! ATTENZIONE !!

Ti sei connesso ad un access point non valido.

Ciò può essere dovuto a due ragioni:

- Se stai usando un dispositivo Android, potresti aver impostato la rete **Studenti** con l'opzione **"Certificato CA"** su **"Non convalidare"**. In questo caso, puoi trovare a questo link una guida su come configurare in modo sicuro la rete sul tuo dispositivo;
- Se il tuo dispositivo è un iPhone o iPad, oppure un Android con il campo **"Certificato CA"** impostato correttamente, potresti aver autonomamente deciso di collegarti a questa rete, nonostante il telefono non si sia collegato in maniera automatica. Stai molto attento le prossime volte, in quanto connettendoti a reti come questa le tue credenziali potrebbero essere rubate da un utente malintenzionato!

Procedi alla pagina web di UniBs con la guida di configurazione

Questa pagina è stata realizzata dall'Università degli Studi di Brescia per verificare la presenza di dispositivi configurati non correttamente dal punto di vista della sicurezza ed avvisare gli utenti in modo che possano correggere la propria configurazione. Le informazioni sono raccolte in forma anonima ed utilizzate solo a fini statistici.

Figure 11: The informative captive portal shown to the users who successfully authenticated on our evil-twins.

8.4 Outcomes

After deploying the evil-twins inside our university, we let them run for some days in order to collect as much data as we could, then we retrieved the log files containing the pieces of information we were interested in. By the time we are writing these results, our three evil-twins have been active for around seven working days.

As we suspected, we were able to try to authenticate a big amount of users, despite the fact that our evil-twins had far less powerful signal with respect to the genuine access points. In total, 295 different users attempted to connect to our network at least once. Among these, no one used "anonymous" (or small variations of it) as anonymous identity; this means that it is highly probable that everyone tried to access the network disclosing their personal identity in the clear. As it turns out, every user from our experiment is vulnerable to tracking attacks, thus confirming our suspicions about the low adoption of anonymous identities by users. Moreover, among the various access attempts, 109 devices (36.95%) did not use random MAC addresses, thus compromising user's privacy once again.

Coming to the users who completed the authentication against our evil-twins, our data show that we were able to authenticate exactly 96 users. This means that 32.54% of the users (around one every three) who tried to connect to the network completed the authentication, thus exposing themselves to harsher threats. Here, the rate of non-randomized MAC addresses is way higher: 64 devices (66.67%) still use the hardware MAC of the network interface. Previously, when talking about access attempts in general, it was said that no user used "anonymous" or some of its variants as anonymous identity. However, we cannot conclude that no anonymous identity was used, even if it is highly likely. Here, having access to both anonymous and real identities, we can affirm that anonymous identities surely are not popular. Among the users who authenticated, only two of them used different values for anonymous identity and personal identity. However, these anonymous identities are different, meaning that they are probably still tied to the user, and are of course different from "anonymous" et similia. Again, virtually every user who authenticated does not use an anonymous identity for sure, thus making us think that even the users who did not completely authenticate were using their real username as anonymous identity. Coming to the authentication methods, all of the 96 users who did authenticate used EAP-PEAP for phase 1 authentication, while no one opted for EAP-TTLS. The phase 2 authentication method set on each smart device is more interesting as attackers, as the difficulty of stealing a password is strongly related to the authentication protocol used. The following table contains, for each phase 2 authentication method recorded in our log files, the amount of supplicants that requested that authentication method.

Phase 2 method	**Amount (abs)**	**Amount (%)**
MS-CHAPv2	50	52.08%
PWD	29	30.21%
GTC	17	17.71%

Table 2: The phase 2 authentication methods chosen by the supplicants during our attack. In the second column, how many times that authentication method has been picked.

As it can be clearly seen, the vast majority of devices uses MS-CHAPv2 for phase 2 authentication, probably because of the fact that many modern Android devices have this method set as default. As already said previously, it is not trivial for an attacker to retrieve the password when the user authenticates with MS-CHAPv2, but it can be done with enough computational capabilities. For this reason, users who authenticated with MS-CHAPv2 are exposed to credential stealing attacks too. On the other hand, it can be seen that some supplicants accepted to use GTC for phase 2 authentication, probably because no phase 2 method was set on the supplicant. These users basically gift their password to the attacker, so they surely are the most endangered ones. Finally, surprisingly enough there were many users who intended to authenticate using PWD. As already said in 3.2, PWD is considered one of the safest authentication methods since it provides mutual authentication and it is based on discrete logarithms, a problem which is hard to solve as of today. These users are the only ones, among those who authenticated, who can be considered safe from having their password stolen.

As predicted during our laboratory experiments, only Android devices authenticated against our evil-twins. This is due to the fact that iOS devices won't connect to an evil-twin without the user looking for Wi-Fi networks and accepting a new certificate, while Android smartphones can do so if misconfigured. Since our evil-twins were placed inside the university, iOS devices could sense both the genuine network and the evil one, preferring the first one as the certificate was already known by the smart device. For misconfigured Android devices, instead, only the signal power matters in order to decide whether to connect to a known network or not. Since while passing near the evil-twin its signal is stronger than that of the genuine network, misconfigured devices opted for the rogue network instead of the genuine one. By analyzing the data concerning the smartphones who connected to our rogue network, we discovered that the devices which fell into the attack belonged to the following brands:

Brand	**Amount (abs)**	**Amount (%)**
Samsung	28	29.17%
Huawei	24	25.00%
Xiaomi	16	16.67%
Other	28	29.17%

Table 3: The brands of the supplicants who successfully authenticated against our evil-twins.

To clarify, the "Other" category comprises devices from various other brands, such as LG, Sony, OPPO, and devices that we were not able to identify. Since for each one of these brands there were at most three devices recorded, we decided to group them in a common category.

9 Defensive mechanisms

After discussing which vulnerabilities can be exploited to carry out several attacks, it is mandatory to show how some of these security problems can be tackled on Android and iOS devices.

Since the two most important vulnerabilities exploit the lack of anonymous identity and the fact that phase 1 authentication can easily succeed, be it for a wrong configuration of the device or for a user action, we will discuss how users can stem these problems. Credential stealing attacks, authentication of the user, traffic sniffing and DNS spoofing all need for the phase 1 authentication to succeed, so if a user manages to avoid stepping further in a conversation with a rogue AS, then he can consider himself safe.

Concerning tracking attacks, as of today only Android users can defend themselves easily. In order to do so, every Android user should not leave the "anonymous identity" field empty, but should, instead, fill it with something like *anonymous@domain.TLD*. To make an example, a user from the University of Brescia, whose domain is "unibs" and whose TLD is "it", should set this field with *anonymous@unibs.it*. Some universities may have policies that restrict the possible values for anonymous identities to a certain set, so in this case users should also ask to the network administrators of their universities for more details. In general, however, the first portion of the anonymous identities can be any string, while the domain must always be the right one. For iOS users, on the other hand, there is no easy solution as of today. The only way to use an anonymous identity is by installing a configuration profile, which can be created with specific programs or provided by the university, for example via eduroam CAT. Unluckily, creating a configuration profile can be not so easy for a user, so it should be up to each university to provide its users with the right configuration profile, containing an anonymous identity. The only other way for an iOS user to minimize the odds of being tracked is to turn off his Wi-Fi when roaming in public. This does not prevent an attacker from tracking the user nearby the university or the user's house, where the Wi-Fi is most likely on, but the attacker's evil-twin has to compete with other Wi-Fi networks to successfully attack the user.

The second major problem, as said before, is that some Android devices do not validate the server certificate, while iOS devices require for a manual verification of it. Again, at least for Android smartphones, universities are the ones responsible for the lack of security in their users' devices. To avoid a big amount of harmful attacks, we encourage every user to ask for more detailed instructions to their university on how to set up the eduroam network. In case a university uses a self-signed certificate, each Android user should be able to find the right root CA certificate on a web page of his university, download it, install it and use it in the "CA certificate" field of the network configuration. In case a university uses a certificate signed by a globally trusted CA, instead, users must be provided with the right domain to look for in the certificate, to make sure that the certificate exposed by the AS is the right one. For Android smartphones in general, the option "Do not validate" must never be used, even if the university says so. On the other hand, iOS devices will always give a user the option to accept a new certificate for eduroam, even by installing a trusted certificate with a configuration profile. For this reason, iOS users in particular should set up the connection to the network inside the university, where attacks are harder to carry out, and then reject any new certificate, unless their university tells otherwise. Since both Android and iOS devices generally give the option to the user to connect anyway, either by tapping a "Try again" button or by accepting a new certificate, universities should teach their users to always step back from networks where something like this occurs.

To summarize, concerning tracking attacks, Android users should always set the anonymous identity so that it is really anonymous. iOS users, who don't have this possibility, can either ask their university for a configuration profile which uses an anonymous identity or create one themselves. In case none of these options is available, turning off the Wi-Fi when away from safe places can limit, but certainly not eliminate, the attacker's odds of succeeding. On the other hand, regarding every other attack, Android users must be able to properly configure their smartphone, avoiding the "Do not validate" option completely. iOS users, along with Android users too, must be careful when something goes wrong with the eduroam network: a new certificate prompted to an iOS user or a connection error on an Android device most certainly mean that the network they tried to connect to is not genuine, so they should step back.

10 Conclusions

In this work, we deeply discussed the most important aspects of 802.1X authentication in WPA-Enterprise networks. As it was briefly said, WPA-Enterprise guarantees, as of today, a higher level of security to end users with respect to WPA-Personal. However, in order for a user to take full advantage of the security mechanisms provided by WPA-Enterprise, he must be fully aware on how to properly configure his smart devices. Even a small error in the configuration process can easily lead to harsh consequences.

In case of the eduroam network, which was the one we developed and tested attacks against, it was shown how the threats to users privacy are generally tied to wrong directives from universities. As a matter of fact, the vast majority of universities do not endorse anonymous identities, thus leaving users exposed to tracking attacks, and some tell their users to skip one of the most important phases in 802.1X authentication, which is server-side authentication.

If universities are the ones that must be blamed for downgrading their users' privacy, Google, Apple and other smart devices manufacturers are also equally guilty. On one hand, Google itself can be declared the safest among the various brands, as Google's Pixels are lacking, since the end of 2020, of the option to skip server-side authentication, thus mandating the user to choose a way to validate the server certificate. Other Android devices, on the other hand, still have the option "Do not validate" available. Every Android ROM developer should be encouraged to follow Google's guidelines, for the sake of users' security. Finally, Apple should automatize the verification of certificates as it happens on Android devices. Prompting a certificate to the end user is not a wise choice, as users are not generally able to distinguish between a fake and a genuine certificate, and it is a task they should never perform.

Among the attacks presented in this work, we can safely say that user tracking is by far the easiest one to perform. What we hope is for universities to start considering this kind of attack as a serious threat to users' privacy. Universities should start adopting anonymous identities as soon as possible, make sure that their users use them and make available configuration profiles for any device, so that both Android and iOS devices can be put at safety.

The results of our experiments are clear: almost every user can easily be tracked in space and time by an attacker because of the low-to-none adoption rate of anonymous identities. The number of different users who successfully authenticated against our evil-twins is not that low, as it can be clearly seen that around one in three students actually authenticated himself; this is not a positive datum for our university or any other one that instructs its users to use the "Do not validate" option. These results should strongly encourage universities in changing their security policies and start using every 802.1X security mechanism at its best, for the sake of users' privacy.

In general, with a mixed action by both universities and smart devices manufacturers, users could eventually make the most out of the security tools provided by WPA-Enterprise. Without the will to change things as they are now, from both perspectives of universities and manufacturers, users will continue to be threatened by many attacks which are easy to set up and really harmful.

11 Acknowledgements

327640385b5e800294e3084d6f51fb9fd6e2d185e104b91826755e461867c8fc

12 Sommario

Le reti Wireless LAN, comunemente chiamate reti Wi-Fi, hanno ormai una lunga storia. Sin dagli albori di queste reti di comunicazione, ingegneri e accademici da svariate branche dell'informatica, dell'elettronica e delle telecomunicazioni hanno tentato di rendere queste reti il più sicure possibile. Per raggiungere questo scopo, sono stati inventati e implementati molti strumenti che mirano a difendere gli utenti della rete da molti possibili attacchi alla loro sicurezza. Uno sforzo così grande si è reso necessario a causa della natura di queste reti, in quanto scambiando informazioni nell'etere sono intrinsecamente meno sicure delle reti cablate.

Tra le varie famiglie di reti Wi-Fi, una si è sempre distinta per l'elevato livello di sicurezza che è in grado di fornire ai suoi utenti, impareggiato almeno fino alle seconda generazione inclusa. Questa famiglia di reti WLAN è generalmente chiamata WPA-Enterprise, WPA-EAP o anche semplicemente 802.1X. Le reti WPA-Enterprise non sono certamente comuni quanto le reti WPA-Personal, che sono ormai presenti in ogni abitazione, ma si trovano molto spesso in contesti dove un elevato numero di utenti deve avere accesso a una stessa rete.

All'interno della famiglia WPA-Enterprise, una particolare rete svetta sulle altre in fatto di diffusione. Si tratta della rete "eduroam", ovvero una rete Wi-Fi supportata da un enorme numero di università in tutto il mondo. Dato che le università possono contare migliaia di studenti, centinaia di docenti e un gran numero di membri del personale in generale, è immediato capire come questa rete abbia un numero di utenti enorme. Purtroppo, un altrettanto grande numero di università continua a istruire in maniera incorretta i suoi utenti su come configurare l'accesso a questa rete, rendendoli così vulnerabili a una più o meno lunga serie di attacchi.

La rete eduroam, così come in generale le reti WPA-Enterprise, ha una lunga storia di attacchi alle spalle. Alcuni di questi attacchi sono ben noti dalla comunità, mentre altri sono o meno noti, o meno considerati o ancora considerati non particolarmente dannosi. Inoltre, durante lo studio e la ricerca di alcuni attacchi a questo tipo di reti, ci siamo resi conto di come per molti di essi manchi sufficiente documentazione per replicarli o per poterli comprendere appieno.

Il nostro obiettivo principale è stato quello di studiare, sviluppare e inscenare vari attacchi agli utenti di questa rete mediante l'utilizzo di uno o più gemelli malvagi. Un gemello malvagio è un dispositivo, nel nostro caso un dispositivo di rete, che replica i comportamenti di un dispositivo genuino, ma con intenzioni malevole. Nel caso in esame, il gemello malvagio ha lo scopo di replicare un access point per la rete eduroam in grado di permettere agli utenti di tale rete di autenticarsi presso di esso, in modo da poter poi effettuare una serie di attacchi che minano alla privacy degli utenti, rubando informazioni di interesse. Forte interesse è stato posto anche nel comprendere sotto quali condizioni un certo dispositivo smart possa cadere in un attacco tra quelli sviluppati e, viceversa, sotto quali ipotesi tale dispositivo può dirsi al sicuro.

Relativamente agli attacchi che abbiamo studiato, sviluppato e implementato, possiamo dire che a seconda della configurazione di un dispositivo smart, la piattaforma di attacco a disposizione dell'attaccante può essere più o meno ampia. Tra gli attacchi che abbiamo testato in laboratorio, i più importanti sono sicuramente:

- Tracciamento dell'utente. Ciò rende possibile a un attaccante sapere con più o meno precisione la posizione di un utente all'interno del campo della rete malevola. A causa di come la rete eduroam viene gestita dalla stragrande maggioranza delle università, praticamente ogni utente è vulnerabile a questo attacco;

- Furto delle credenziali di accesso. Questo attacco è più difficile del precedente da attuare, ma ancora una volta è permesso in genere da errate istruzioni delle università relativamente la configurazione della rete. Se, inoltre, l'utente non diversifica le password, la password rubata potrebbe garantire all'attaccante l'accesso ad altri servizi;

- De-anonimizzazione del dispositivo. Se l'utente si autentica presso la rete malevola, l'attaccante può ottenere un gran numero di informazioni relative al dispositivo utilizzato, che possono poi permettergli di inscenare altri attacchi che sfruttano vulnerabilità di certi dispositivi o versioni dei sistemi operativi;

- Sniffing del traffico. Dato che il dispositivo smart utilizza l'evil-twin come punto di accesso al web, tutto il traffico internet che il dispositivo invia o riceve deve passare per l'evil-twin, che quindi ha accesso a queste informazioni. L'attaccante può quindi vedere quali siti vengono visitati dall'utente, spesso quali applicazioni ha installato sul suo dispositivo e, in rari casi, anche quali dati scambia con i siti web;

- Spoofing del DNS. Dato che il traffico passa per il gemello malvagio, il dispositivo dell'utente si rivolgerà ad esso per la risoluzione di nomi simbolici in indirizzi IP, utilizzando il gemello malvagio come resolver DNS. Ciò implica che in alcuni casi l'attaccante può dirottare le richieste per un certo sito web su di un altro sito, magari controllato dall'attaccante stesso, che può quindi procedere con un attacco di tipo man-in-the-middle o phishing. Come spiegato nella sezione dedicata, oggi questi attacchi sono più difficili da inscenare grazie all'elevata adozione del protocollo HTTPS rispetto ad HTTP. In ogni caso, esistono browser per smartphone che utilizzano di default HTTP invece di HTTPS, rendendo questi attacchi molto pericolosi per alcuni utenti.

In generale, tolto il primo attacco tra quelli elencati, tutti gli altri richiedono che l'utente abbia configurato in maniera marcatamente errata lo smartphone oppure che si sia volontariamente connesso alla rete, nonostante gli errori di connessione o gli avvertimenti presentati dallo smartphone.

I dispositivi in questione possono essere divisi in due ampie classi, ovvero i dispositivi Android e i dispositivi iOS. I primi sono generalmente più propensi a cadere nei vari attacchi, in quanto lasciano più libertà di configurazione all'utente, spesso presentando opzioni di default molto insicure. Google ha imposto che, a partire da Dicembre 2020, alcune opzioni di configurazione debbano essere rimosse in quanto possono degradare eccessivamente la sicurezza degli utenti sulle reti 802.1X; purtroppo però, ad oggi ancora moltissimi dispositivi Android possono essere configurati con queste opzioni. I dispositivi iOS, invece, lasciano molta meno libertà in fase di configurazione e si sono dimostrati più resistenti ad alcuni attacchi che abbiamo sviluppato. In ogni caso, nei dispositivi Apple più che in quelli Android, se l'utente decide attivamente di collegarsi alla rete malevola, confondendola con quella genuina, può sottoporsi a danni maggiori rispetto a un dispositivo Android. In linea di massima, possiamo affermare che i dispositivi Android possano fornire maggiore sicurezza rispetto a quelli iOS ma, nonostante ciò, risultino più insicuri sia a causa di errori degli utenti, sia a causa delle università che istruiscono in maniera errata gli utenti relativamente la configurazione della rete. I dispositivi iOS, invece, risultano meno vulnerabili, ma comunque non completamente sicuri a causa di vincoli imposti dal sistema operativo e, soprattutto, fortemente indeboliti da semplici errori dell'utente.

Per meglio comprendere quanto questi attacchi siano fattibili in pratica, al di fuori di test controllati in laboratorio, abbiamo organizzato un esperimento sul campo, con l'obiettivo di capire quanti utenti reali potessero cadere nelle varie trappole predisposte. Effettuare un test live è di fondamentale importanza per capire quanti utenti effettivamente siano esposti alle minacce che abbiamo studiato, così da darci una misura di applicabilità ed efficacia di tali attacchi. L'altro scopo di punta dell'esperimento, oltre a raccogliere dati dagli utenti, era quello di avvertirli qualora questi si fossero effettivamente connessi alla rete malevola e di spiegare loro come risolvere questo importante inconveniente. Ciò è stato possibile tramite la realizzazione di un captive portal informativo, che spiega all'utente le possibili ragioni per cui può essere caduto nella trappola e indica a questo delle guide per poter risolvere questi problemi. Il captive portal realizzato, visionabile in figura 11 (paragrafo 8.3.4), è scritto interamente in italiano in quanto gli utenti sotto attacco erano principalmente studenti della nostra università. L'esperimento che abbiamo svolto sul campo è stato concordato in ogni aspetto con il team ICT della nostra università, il quale ci ha dato tutti i permessi per svolgere il nostro test. Su loro consiglio, abbiamo deciso di attaccare la rete interna della nostra università, chiamata "Studenti", invece della rete eduroam. Questa scelta è stata presa considerando i seguenti aspetti:

- Sarebbe stato opportuno evitare di allarmare studenti stranieri, i quali avrebbero contattato le loro università per avere delucidazioni. Queste, a loro volta, non sarebbero state a conoscenza dell'esperimento e quindi avrebbero potuto segnalare la cosa ad autorità competenti;
- La nostra università vanta un numero piuttosto basso di ospiti, quindi gli utenti della rete eduroam sono praticamente gli stessi della rete Studenti;
- Gli studenti Erasmus presso la nostra università vengono spesso forniti di credenziali legate strettamente alla nostra università, quindi tendono ad utilizzare la rete Studenti piuttosto che eduroam;
- Stando ai dati real-time che ci sono stati forniti dal team ICT, gli utenti connessi alla rete Studenti sono circa dieci volte di più rispetto a quelli connessi a eduroam. Ciò significa che attaccando la rete Studenti avremmo potuto ottenere un quantitativo molto più grande di dati.

Per effettuare questo test, abbiamo configurato tre Raspberry Pi 3 Model B+ con alcune antenne Wi-Fi (visionabili in figura 10, paragrafo 8.2) e con tutti i software necessari per ricreare la rete Wi-Fi in esame. I software utilizzati sono stati modificati in maniera più o meno ampia in modo da permetterci di raggiungere gli obiettivi prefissati. Una volta pronte e ricevute tutte le autorizzazioni da parte degli enti competenti della nostra università, abbiamo provveduto a nascondere le tre Raspberry in punti diversi della nostra università, in modo da raccogliere quanti più dati possibili. I dati sono stati anonimizzati in modo da non poter risalire agli utenti reali; per anonimizzare le identità degli utenti, abbiamo utilizzato la funzione di hashing SHA256, che ad oggi è considerata fortemente sicura. A fine esperimento, abbiamo raccolto e analizzato i file di log per poter estrarre le informazioni d'interesse. Al momento della stesura di questo documento, i nostri gemelli malvagi sono stati operativi per circa sette giorni lavorativi.

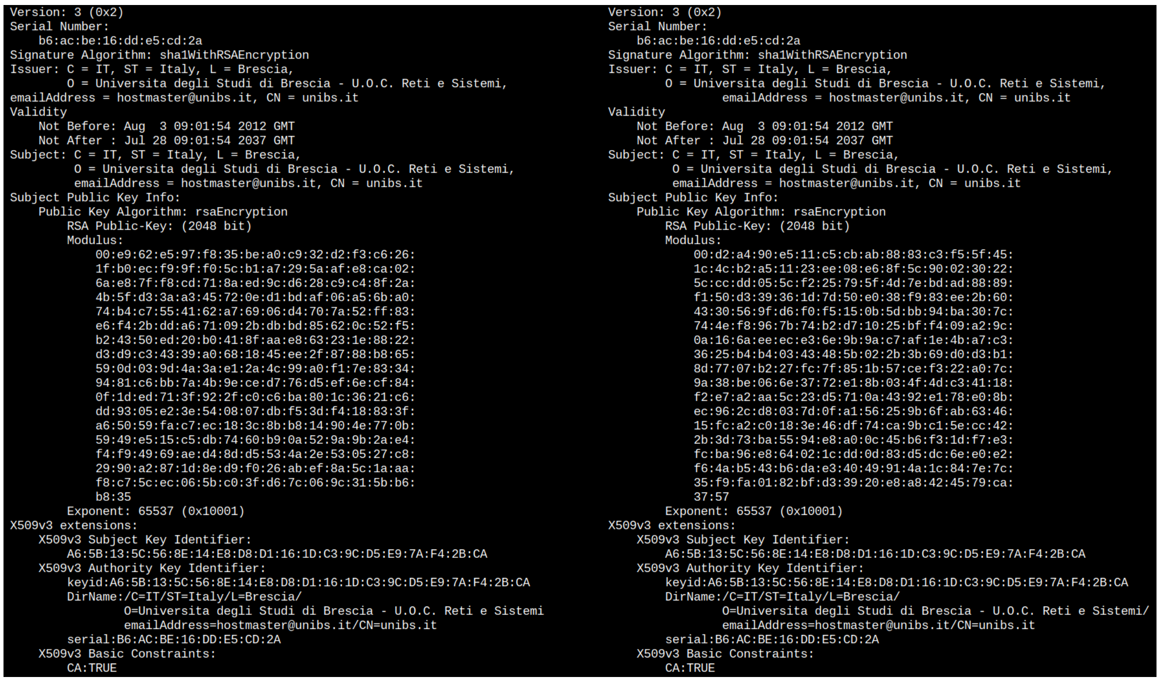
Stando ai risultati del nostro esperimento, abbiamo potuto notare come la stragrande maggioranza degli utenti non faccia uso di identità anonima, esponendosi quindi ad attacchi di localizzazione. In particolare, su 296 tentativi di accesso alla rete da utenti distinti, nessun è stato eseguiti con identità anonima uguale a "anonymous" o sue comuni varianti (e.g. "Anonymous", "ANONYMOUS", "anonimo", ...). Possiamo quindi immaginare senza troppa fantasia che le identità scambiate in chiaro dagli utenti siano proprio i loro username. Per tutti questi utenti, quindi, risulta banale per un attaccante conoscere la posizione dell'utente in un certo istante di tempo. Inoltre, è interessante notare come una percentuale piuttosto alta dei tentativi di accesso, nello specifico il 36.95%, sia avvenuto tramite dispositivi con indirizzo MAC non casuale; tali dispositivi sono per natura tracciabili con facilità. Per quanto riguarda i tentativi di autenticazione che hanno avuto successo, invece, ne abbiamo registrati 96 in totale (32.54% sul totale, circa un utente ogni tre). Se prima non era possibile dire con certezza se gli utenti utilizzassero un'identità anonima, qui è possibile in quanto disponiamo di dati in più. Per ogni run di autenticazione, infatti, possediamo, per ogni dispositivo, sia l'hash dell'identità anonima, sia l'hash dell'identità vera. Di ogni autenticazione compiuta, solo 2 su 96 riportano due hash distinti per identità anonima e personale; entrambe le identità anonime sono però differenti, il che ci fa pensare che siano comunque in qualche modo collegate all'utente, e sono ovviamente diverse da "anonymous" et similia. Ciò ci porta quindi a pensare che anche gli utenti non autenticati in realtà usino la loro identità personale al posto di una anonima. Parlando di metodi di autenticazione scelti, tutte le autenticazioni sono avvenute utilizzando EAP-PEAP come metodo di fase 1. Il metodo di fase 1 non è di cruciale importanza per un attaccante, bensì è il metodo di fase 2 ad essere più interessante in quanto la facilità con cui l'attaccante può rubare la password dell'utente è fortemente dipendente da questo metodo. Nel caso del nostro esperimento, in tabella 2 (paragrafo 8.4) sono riportati i metodi di autenticazione di fase 2 richiesti dai dispositivi degli utenti congiuntamente al numero di volte che sono stati richiesti. Nel caso del metodo GTC, cioè quello che viene proposto al dispositivo da parte del nostro gemello malvagio, la difficoltà nel rubare la password è praticamente nulla, in quanto questa viene consegnata in chiaro. Il metodo MS-CHAPv2, invece, è sicuramente più resistente e rende il recupero della password più difficile, ma ad oggi ampiamente fattibile. Infine, il metodo PWD è sicuramente il più resistente in termini di furto della password, in quanto si basa sugli stessi principi matematici su cui è basato lo scambio di chiave Diffie-Hellman (difficoltà nel risolvere i logaritmi discreti). Stando al nostro esperimento, dunque, possiamo affermare che una quantità molto consistente di utenti è soggetta al tracciamento nello spazio, a causa del mancato utilizzo di un'identità identità anonima. Per quanto riguarda gli utenti che si sono autenticati completamente presso i nostri gemelli malvagi, è possibile notare come questi siano una frazione abbastanza consistente di tutti quelli che hanno tentato la connessione (come detto prima, circa un utente su tre). Per questo motivo, il fatto che un numero non trascurabile di utenti si sia connesso, esponendosi quindi a una serie di attacchi dannosi per la loro sicurezza, non è sicuramente un dato incoraggiante per la nostra università.

Bisogna augurarsi che, in futuro, tutte le università del mondo inizino a istruire gli utenti a utilizzare le identità anonimizzate per l'autenticazione alla rete eduroam o alle reti 802.1X di ateneo in generale, in quanto ad oggi praticamente ogni utente di ogni università può essere localizzato nello spazio da un attaccante, proprio a causa del mancato utilizzo di questo strumento. Inoltre, è anche necessario che le università dedichino più attenzione ad istruire tutti gli utenti, al di là del livello di esperienza, su come configurare correttamente le connessioni alla reti sui propri telefoni. Ad oggi, infatti, ancora molte guide universitarie riportano informazioni errate e fortemente dannose per la privacy utenti.

13 Appendix A - CA certificates

In this section, we will show the root CA certificate of our university and compare it with the fake one we crafted. Our purpose was to make the rogue one look as similar as the genuine one, in order to possibly trick users into accepting ours instead of the right one.

The following figure shows how similar a root CA certificate and a counterfeit one can be. This picture shows every possible field but the signature and the certificate itself, which will be shown in another picture. As it can be seen, in this picture only the modulus of the public key is different, while every other field is identical in both certificates.

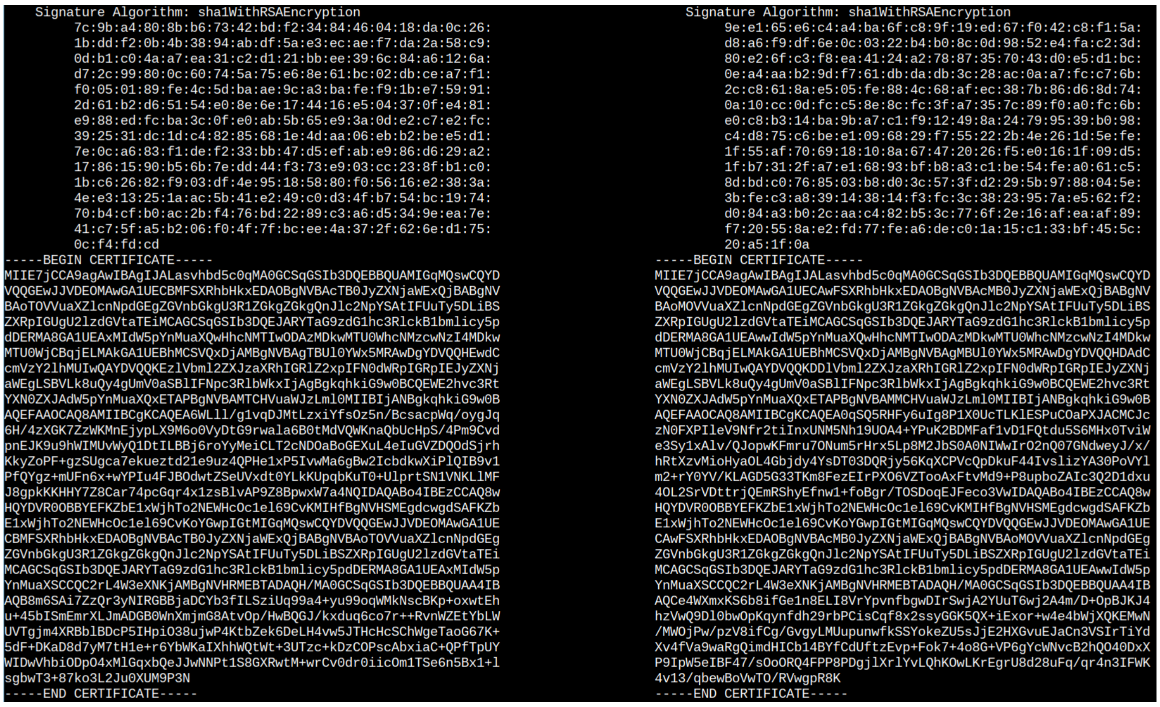


```
Version: 3 (0x2)
Serial Number:
    b6:ac:be:16:dd:e5:cd:2a
Signature Algorithm: sha1WithRSAEncryption
Issuer: C = IT, ST = Italy, L = Brescia,
        O = Universita degli Studi di Brescia - U.O.C. Reti e Sistemi,
emailAddress = hostmaster@unibs.it, CN = unibs.it
Validity
    Not Before: Aug  3 09:01:54 2012 GMT
    Not After : Jul 28 09:01:54 2037 GMT
Subject: C = IT, ST = Italy, L = Brescia,
         O = Universita degli Studi di Brescia - U.O.C. Reti e Sistemi,
         emailAddress = hostmaster@unibs.it, CN = unibs.it
Subject Public Key Info:
    Public Key Algorithm: rsaEncryption
        RSA Public-Key: (2048 bit)
        Modulus:
            00:e9:62:e5:97:f8:35:be:a0:c9:32:d2:f3:c6:26:
            1f:b0:ec:f9:9f:f0:5c:b1:a7:29:5a:af:e8:ca:02:
            6a:e8:7f:f8:cd:71:8a:ed:9c:d6:28:c9:c4:8f:2a:
            4b:5f:d3:3a:a3:45:72:0e:d1:bd:af:06:a5:6b:a0:
            74:b4:c7:55:41:62:a7:69:06:d4:70:7a:52:ff:83:
            e6:f4:2b:dd:a6:71:09:2b:db:bd:85:62:0c:52:f5:
            b2:43:50:ed:20:b0:41:8f:aa:e8:63:23:1e:88:22:
            d3:d9:c3:43:39:a0:68:18:45:ee:2f:87:88:b8:65:
            59:0d:03:9d:4a:3a:e1:2a:4c:99:a0:f1:7e:83:34:
            94:81:c6:bb:7a:4b:9e:ce:d7:76:d5:ef:6e:cf:84:
            0f:1d:ed:71:3f:92:2f:c0:c6:ba:80:1c:36:21:c6:
            dd:93:05:e2:3e:54:08:07:db:f5:3d:f4:18:83:3f:
            a6:50:59:fa:c7:ec:18:3c:8b:b8:14:90:4e:77:0b:
            59:49:e5:15:c5:db:74:60:b9:0a:52:9a:9b:2a:e4:
            f4:f9:49:69:ae:d4:8d:d5:53:4a:2e:53:05:27:c8:
            29:90:a2:87:1d:8e:d9:f0:26:ab:ef:8a:5c:1a:aa:
            f8:c7:5c:ec:06:5b:c0:3f:d6:7c:06:9c:31:5b:b6:
            b8:35
        Exponent: 65537 (0x10001)
X509v3 extensions:
    X509v3 Subject Key Identifier:
        A6:5B:13:5C:56:8E:14:E8:D8:D1:16:1D:C3:9C:D5:E9:7A:F4:2B:CA
    X509v3 Authority Key Identifier:
        keyid:A6:5B:13:5C:56:8E:14:E8:D8:D1:16:1D:C3:9C:D5:E9:7A:F4:2B:CA
        DirName:/C=IT/ST=Italy/L=Brescia/
                O=Universita degli Studi di Brescia - U.O.C. Reti e Sistemi
                emailAddress=hostmaster@unibs.it/CN=unibs.it
        serial:B6:AC:BE:16:DD:E5:CD:2A
    X509v3 Basic Constraints:
        CA:TRUE
```

```
Version: 3 (0x2)
Serial Number:
    b6:ac:be:16:dd:e5:cd:2a
Signature Algorithm: sha1WithRSAEncryption
Issuer: C = IT, ST = Italy, L = Brescia,
        O = Universita degli Studi di Brescia - U.O.C. Reti e Sistemi,
                emailAddress = hostmaster@unibs.it, CN = unibs.it
Validity
    Not Before: Aug  3 09:01:54 2012 GMT
    Not After : Jul 28 09:01:54 2037 GMT
Subject: C = IT, ST = Italy, L = Brescia,
         O = Universita degli Studi di Brescia - U.O.C. Reti e Sistemi,
         emailAddress = hostmaster@unibs.it, CN = unibs.it
Subject Public Key Info:
    Public Key Algorithm: rsaEncryption
        RSA Public-Key: (2048 bit)
        Modulus:
            00:d2:a4:90:e5:11:c5:cb:ab:88:83:c3:f5:5f:45:
            1c:4c:b2:a5:11:23:ee:08:e6:8f:5c:90:02:30:22:
            5c:cc:dd:05:5c:f2:25:79:5f:4d:7e:bd:ad:88:89:
            f1:50:d3:39:36:1d:7d:50:e0:38:f9:83:ee:2b:60:
            43:30:56:9f:d6:f0:f5:15:0b:5d:bb:94:ba:30:7c:
            74:4e:f8:96:7b:74:b2:d7:10:25:bf:f4:09:a2:9c:
            0a:16:6a:ee:ec:e3:6e:9b:9a:c7:af:1e:4b:a7:c3:
            36:25:b4:b4:03:43:48:5b:02:2b:3b:69:d0:d3:b1:
            8d:77:07:b2:27:fc:7f:85:1b:57:ce:f3:22:a0:7c:
            9a:38:be:06:6e:37:72:e1:8b:03:4f:4d:c3:41:18:
            f2:e7:a2:aa:5c:23:d5:71:0a:43:92:e1:78:e0:8b:
            ec:96:2c:d8:03:7d:0f:a1:56:25:9b:6f:ab:63:46:
            15:fc:a2:c0:18:3e:46:df:74:ca:9b:c1:5e:cc:42:
            2b:3d:73:ba:55:94:e8:a0:0c:45:b6:f3:1d:f7:e3:
            fc:ba:96:e8:64:02:1c:dd:0d:83:d5:dc:6e:e0:e2:
            f6:4a:b5:43:b6:da:e3:40:49:91:4a:1c:84:7e:7c:
            35:f9:fa:01:82:bf:d3:39:20:e8:a8:42:45:79:ca:
            37:57
        Exponent: 65537 (0x10001)
X509v3 extensions:
    X509v3 Subject Key Identifier:
        A6:5B:13:5C:56:8E:14:E8:D8:D1:16:1D:C3:9C:D5:E9:7A:F4:2B:CA
    X509v3 Authority Key Identifier:
        keyid:A6:5B:13:5C:56:8E:14:E8:D8:D1:16:1D:C3:9C:D5:E9:7A:F4:2B:CA
        DirName:/C=IT/ST=Italy/L=Brescia/
                O=Universita degli Studi di Brescia - U.O.C. Reti e Sistemi/
                emailAddress=hostmaster@unibs.it/CN=unibs.it
        serial:B6:AC:BE:16:DD:E5:CD:2A
    X509v3 Basic Constraints:
        CA:TRUE
```

Figure 12: On the left, data contained in the genuine UniBs root CA certificate. On the right, data contained in the counterfeit root CA certificate.

For the sake of completeness, the following picture shows the two fields missing in the previous one: the signature and the certificate itself, encoded in Base64. Since the public key of our root CA is different from the original one, the private key is different too. Because of the difference of both public and private keys, the signature is different, as a different public key was signed with a different private key. The certificate itself is different too, as it contains a different signature and a different public key.



```
    Signature Algorithm: sha1WithRSAEncryption
         7c:9b:a4:80:8b:b6:73:42:bd:f2:34:84:46:04:18:da:0c:26:
         1b:dd:f2:0b:4b:38:94:ab:df:5a:e3:ec:ae:f7:da:2a:58:c9:
         0d:b1:c0:4a:a7:ea:31:c2:d1:21:bb:ee:39:6c:84:a6:12:6a:
         d7:2c:99:80:0c:60:74:5a:75:e6:8e:61:bc:02:db:ce:a7:f1:
         f0:05:01:89:fe:4c:5d:ba:ae:9c:a3:ba:fe:f9:1b:e7:59:91:
         2d:61:b2:d6:51:54:e0:8e:6e:17:44:16:e5:04:37:0f:e4:81:
         e9:88:ed:fc:ba:3c:0f:e0:ab:5b:65:e9:3a:0d:e2:c7:e2:fc:
         39:25:31:dc:1d:c4:82:85:68:1e:4d:aa:06:eb:b2:be:e5:d1:
         7e:0c:a6:83:f1:de:f2:33:bb:47:d5:ef:ab:e9:86:d6:29:a2:
         17:86:15:90:b5:6b:7e:dd:44:f3:73:e9:03:cc:23:8f:b1:c0:
         1b:c6:26:82:f9:03:df:4e:95:18:58:80:f0:56:16:e2:38:3a:
         4e:e3:13:25:1a:ac:5b:41:e2:49:c0:d3:4f:b7:54:bc:19:74:
         70:b4:cf:b0:ac:2b:f4:76:bd:22:89:c3:a6:d5:34:9e:ea:7e:
         41:c7:5f:a5:b2:06:f0:4f:7f:bc:ee:4a:37:2f:62:6e:d1:75:
         0c:f4:fd:cd
-----BEGIN CERTIFICATE-----
MIIE7jCCA9agAwIBAgIJALasvhbd5c0qMA0GCSqGSIb3DQEBBQUAMIGqMQswCQYD
VQQGEwJJVDEOMAwGA1UECBMFSXRhbHkxEDAOBgNVBAcTB0JyZXNjaWExQjBABgNV
BAoTOVVuaXZlcnNpdGEgZGVnbGkgU3R1ZGkgZGkgQnJlc2NpYSAtIFUuTy5DLiBS
ZXRpIGUgU2lzdGVtaTEiMCAGCSqGSIb3DQEJARYTaG9zdG1hc3RlckB1bmlicy5p
dDERMA8GA1UEAxMIdW5pYnMuaXQwHhcNMTIwODAzMDkwMTU0WhcNMzcwNzI4MDkw
MTU0WjCBqjELMAkGA1UEBhMCSVQxDjAMBgNVBAgTBUl0YWx5MRAwDgYDVQQHEwdC
cmVzY2lhMUIwQAYDVQQKEzlVbml2ZXJzaXRhIGRlZ2xpIFN0dWRpIGRpIEJyZXNj
aWEgLSBVLk8uQy4gUmV0aSBlIFNpc3RlbWkxIjAgBgkqhkiG9w0BCQEWE2hvc3Rt
YXN0ZXJAdW5pYnMuaXQxETAPBgNVBAMTCHVuaWJzLml0MIIBIjANBgkqhkiG9w0B
AQEFAAOCAQ8AMIIBCgKCAQEA6WLll/g1vqDJMtLzxiYfsOz5n/BcsacpWq/oygJq
6H/4zXGK7ZzWKMnEjypLX9M6o0VyDtG9rwala6B0tMdVQWKnaQbUcHpS/4Pm9Cvd
pnEJK9u9hWIMUvWyQ1DtILBBj6roYyMeiCLT2cNDOaBoGEXuL4eIuGVZDQOdSjrh
KkyZoPF+gzSUgca7ekueztd21e9uz4QPHe1xP5IvwMa6gBw2IcbdkwXiPlQIB9v1
PfQYgz+mUFn6x+wYPIu4FJBOdwtZSeUVxdt0YLkKUpqbKuT0+UlprtSN1VNKLlMF
J8gpkKKHHY7Z8Car74pcGqr4x1zsBlvAP9Z8BpwxW7a4NQIDAQABo4IBEzCCAQ8w
HQYDVR0OBBYEFKZbE1xWjhTo2NEWHcOc1el69CvKMIHfBgNVHSMEgdcwgdSAFKZb
E1xWjhTo2NEWHcOc1el69CvKoYGwpIGtMIGqMQswCQYDVQQGEwJJVDEOMAwGA1UE
CBMFSXRhbHkxEDAOBgNVBAcTB0JyZXNjaWExQjBABgNVBAoTOVVuaXZlcnNpdGEg
ZGVnbGkgU3R1ZGkgZGkgQnJlc2NpYSAtIFUuTy5DLiBSZXRpIGUgU2lzdGVtaTEi
MCAGCSqGSIb3DQEJARYTaG9zdG1hc3RlckB1bmlicy5pdDERMA8GA1UEAxMIdW5p
YnMuaXSCCQC2rL4W3eXNKjAMBgNVHRMEBTADAQH/MA0GCSqGSIb3DQEBBQUAA4IB
AQB8m6SAi7ZzQr3yNIRGBBjaDCYb3fILSziUq99a4+yu99oqWMkNscBKp+oxwtEh
u+45bISmEmrXLJmADGB0WnXmjmG8AtvOp/HwBQGJ/kxduq6co7r++RvnWZEtYbLW
UVTgjm4XRBblBDcP5IHpiO38ujwP4KtbZek6DeLH4vw5JTHcHcSChWgeTaoG67K+
5dF+DKaD8d7yM7tH1e+r6YbWKaIXhhWQtWt+3UTzc+kDzCOPscAbxiaC+QPfTpUY
WIDwVhbiODpO4xMlGqxbQeJJwNNPt1S8GXRwtM+wrCv0dr0iicOm1TSe6n5Bx1+l
sgbwT3+87ko3L2Ju0XUM9P3N
-----END CERTIFICATE-----
```

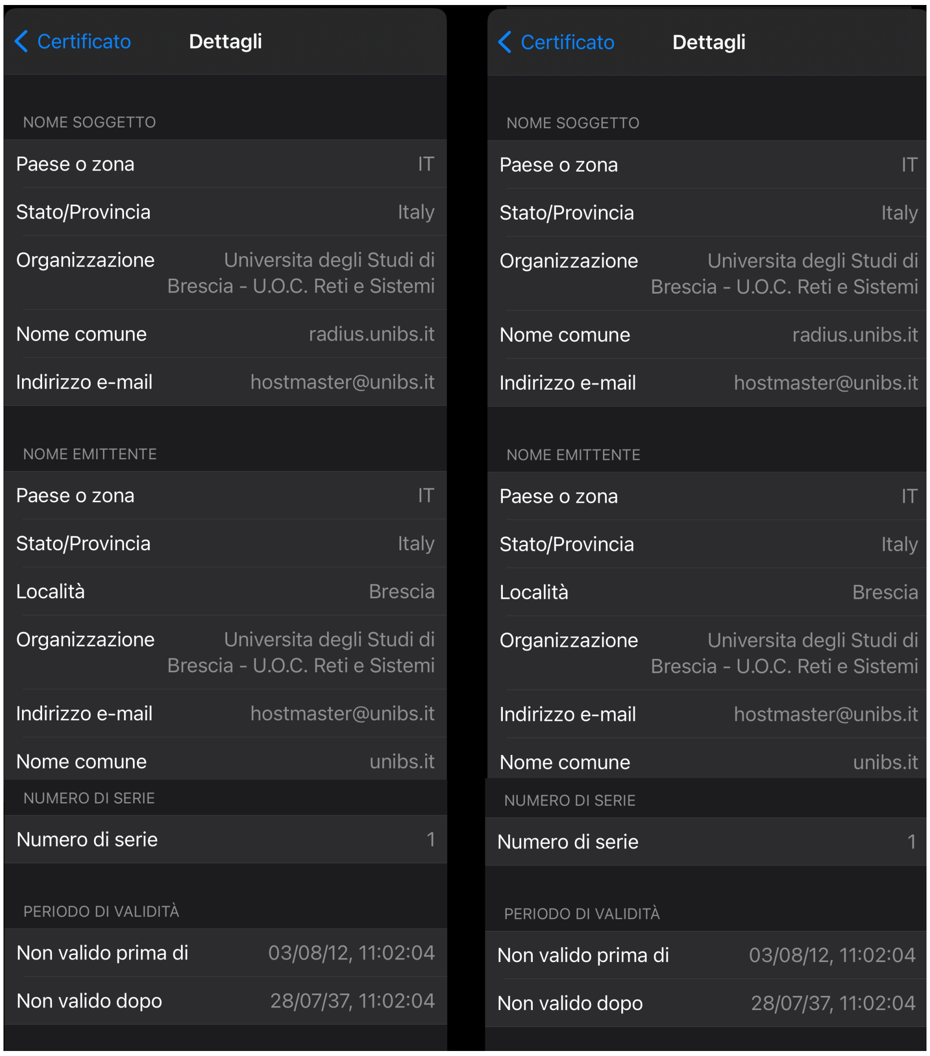
```
    Signature Algorithm: sha1WithRSAEncryption
         9e:e1:65:e6:c4:a4:ba:6f:c8:9f:19:ed:67:f0:42:c8:f1:5a:
         d8:a6:f9:df:6e:0c:03:22:b4:b0:8c:0d:98:52:e4:fa:c2:3d:
         80:e2:6f:c3:f8:ea:41:24:a2:78:87:35:70:43:d0:e5:d1:bc:
         0e:a4:aa:b2:9d:f7:61:db:da:db:3c:28:ac:0a:a7:fc:c7:6b:
         2c:c8:61:8a:e5:05:fe:88:4c:68:af:ec:38:7b:86:d6:8d:74:
         0a:10:cc:0d:fc:c5:8e:8c:fc:3f:a7:35:7c:89:f0:a0:fc:6b:
         e0:c8:b3:14:ba:9b:a7:c1:f9:12:49:8a:24:79:95:39:b0:98:
         c4:d8:75:c6:be:e1:09:68:29:f7:55:22:2b:4e:26:1d:5e:fe:
         1f:55:af:70:69:18:10:8a:67:47:20:26:f5:e0:16:1f:09:d5:
         1f:b7:31:2f:a7:e1:68:93:bf:b8:a3:c1:be:54:fe:a0:61:c5:
         8d:bd:c0:76:85:03:b8:d0:3c:57:3f:d2:29:5b:97:88:04:5e:
         3b:fe:c3:a8:39:14:38:14:f3:fc:3c:38:23:95:7a:e5:62:f2:
         d0:84:a3:b0:2c:aa:c4:82:b5:3c:77:6f:2e:16:af:ea:af:89:
         f7:20:55:8a:e2:fd:77:fe:a6:de:c0:1a:15:c1:33:bf:45:5c:
         20:a5:1f:0a
-----BEGIN CERTIFICATE-----
MIIE7jCCA9agAwIBAgIJALasvhbd5c0qMA0GCSqGSIb3DQEBBQUAMIGqMQswCQYD
VQQGEwJJVDEOMAwGA1UECAwFSXRhbHkxEDAOBgNVBAcMB0JyZXNjaWExQjBABgNV
BAoMOVVuaXZlcnNpdGEgZGVnbGkgU3R1ZGkgZGkgQnJlc2NpYSAtIFUuTy5DLiBS
ZXRpIGUgU2lzdGVtaTEiMCAGCSqGSIb3DQEJARYTaG9zdG1hc3RlckB1bmlicy5p
dDERMA8GA1UEAwwIdW5pYnMuaXQwHhcNMTIwODAzMDkwMTU0WhcNMzcwNzI4MDkw
MTU0WjCBqjELMAkGA1UEBhMCSVQxDjAMBgNVBAgMBUl0YWx5MRAwDgYDVQQHDAdC
cmVzY2lhMUIwQAYDVQQKDDlVbml2ZXJzaXRhIGRlZ2xpIFN0dWRpIGRpIEJyZXNj
aWEgLSBVLk8uQy4gUmV0aSBlIFNpc3RlbWkxIjAgBgkqhkiG9w0BCQEWE2hvc3Rt
YXN0ZXJAdW5pYnMuaXQxETAPBgNVBAMMCHVuaWJzLml0MIIBIjANBgkqhkiG9w0B
AQEFAAOCAQ8AMIIBCgKCAQEA0qSQ5RHFy6uIg8P1X0UcTLKlESPuCOaPXJACMCJc
zN0FXPIleV9Nfr2tiInxUNM5Nh19UOA4+YPuK2BDMFaf1vD1FQtdu5S6MHx0TviW
e3Sy1xAlv/QJopwKFmru7ONum5rHrx5Lp8M2JbS0A0NIWwIrO2nQ07GNdweyJ/x/
hRtXzvMioHyaOL4Gbjdy4YsDT03DQRjy56KqXCPVcQpDkuF44IvslizYA30PoVYl
m2+rY0YV/KLAGD5G33TKm8FezEIrPXO6VZTooAxFtvMd9+P8upboZAIc3Q2D1dxu
4OL2SrVDttrjQEmRShyEfnw1+foBgr/TOSDoqEJFeco3VwIDAQABo4IBEzCCAQ8w
HQYDVR0OBBYEFKZbE1xWjhTo2NEWHcOc1el69CvKMIHfBgNVHSMEgdcwgdSAFKZb
E1xWjhTo2NEWHcOc1el69CvKoYGwpIGtMIGqMQswCQYDVQQGEwJJVDEOMAwGA1UE
CAwFSXRhbHkxEDAOBgNVBAcMB0JyZXNjaWExQjBABgNVBAoMOVVuaXZlcnNpdGEg
ZGVnbGkgU3R1ZGkgZGkgQnJlc2NpYSAtIFUuTy5DLiBSZXRpIGUgU2lzdGVtaTEi
MCAGCSqGSIb3DQEJARYTaG9zdG1hc3RlckB1bmlicy5pdDERMA8GA1UEAwwIdW5p
YnMuaXSCCQC2rL4W3eXNKjAMBgNVHRMEBTADAQH/MA0GCSqGSIb3DQEBBQUAA4IB
AQCe4WXmxKS6b8ifGe1n8ELI8VrYpvnfbgwDIrSwjA2YUuT6wj2A4m/D+OpBJKJ4
hzVwQ9Dl0bwOpKqynfdh29rbPCisCqf8x2ssyGGK5QX+iExor+w4e4bWjXQKEMwN
/MWOjPw/pzV8ifCg/GvgyLMUupunwfkSSYokeZU5sJjE2HXGvuEJaCn3VSIrTiYd
Xv4fVa9waRgQimdHICb14BYfCdUftzEvp+Fok7+4o8G+VP6gYcWNvcB2hQO40DxX
P9IpW5eIBF47/sOoORQ4FPP8PDgjlXrlYvLQhKOwLKrEgrU8d28uFq/qr4n3IFWK
4v13/qbewBoVwTO/RVwgpR8K
-----END CERTIFICATE-----
```

Figure 13: On the left, signature and certificate of the genuine root CA. On the right, signature and certificate of the forged one.

14 Appendix B - Server certificates

In this section, similarly to the previous one, the certificate of the real eduroam hotspot of our university will be presented along with the one we created. A smart device that receives a counterfeit certificate should be able to validate it and fail, as it was not signed with the right private key. However, if a user has the chance to take a look at the certificate itself, as it happens on iOS devices, he may not notice the differences.

This first pictures portrays what the user sees after inspecting the server certificate's details. This screenshots collage shows, side by side, the first fields of the real UniBs RADIUS server certificate and the ones contained in the fake certificate, residing on our evil-twin.



Certificato Dettagli

NOME SOGGETTO

Paese o zona	IT
Stato/Provincia	Italy
Organizzazione	Universita degli Studi di Brescia - U.O.C. Reti e Sistemi
Nome comune	radius.unibs.it
Indirizzo e-mail	hostmaster@unibs.it

NOME EMITTENTE

Paese o zona	IT
Stato/Provincia	Italy
Località	Brescia
Organizzazione	Universita degli Studi di Brescia - U.O.C. Reti e Sistemi
Indirizzo e-mail	hostmaster@unibs.it
Nome comune	unibs.it

NUMERO DI SERIE

Numero di serie	1

PERIODO DI VALIDITÀ

Non valido prima di	03/08/12, 11:02:04
Non valido dopo	28/07/37, 11:02:04

Certificato Dettagli

NOME SOGGETTO

Paese o zona	IT
Stato/Provincia	Italy
Organizzazione	Universita degli Studi di Brescia - U.O.C. Reti e Sistemi
Nome comune	radius.unibs.it
Indirizzo e-mail	hostmaster@unibs.it

NOME EMITTENTE

Paese o zona	IT
Stato/Provincia	Italy
Località	Brescia
Organizzazione	Universita degli Studi di Brescia - U.O.C. Reti e Sistemi
Indirizzo e-mail	hostmaster@unibs.it
Nome comune	unibs.it

NUMERO DI SERIE

Numero di serie	1

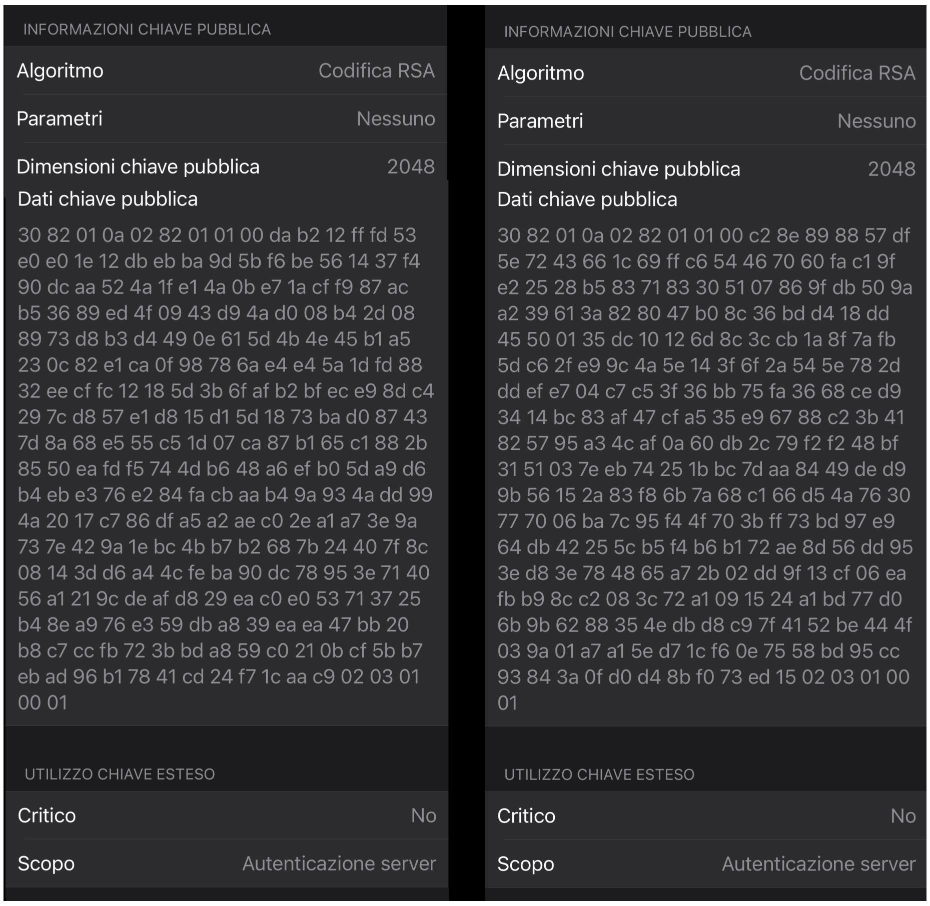
PERIODO DI VALIDITÀ

Non valido prima di	03/08/12, 11:02:04
Non valido dopo	28/07/37, 11:02:04

Figure 14: On the left, identification data contained in the real RADIUS server certificate. On the right, the data contained in the counterfeit one.

As it can be easily seen, this first portion of both certificates is identical, meaning that by looking at these fields a user cannot distinguish between the right certificate and the wrong one. For this reason, even by taking a look at the details of the certificate, a user belonging to the University of Brescia could be easily tempted to accept the fake certificate mistaking it for the real one.

The differences between the two certificates reside, once again, in the RSA public key and, of course, in the signature of the certificate, for the same reasons explained in 13. The following picture portrays the remaining fields of the certificate as it is shown to the user.



INFORMAZIONI CHIAVE PUBBLICA

Algoritmo	Codifica RSA
Parametri	Nessuno
Dimensioni chiave pubblica	2048

Dati chiave pubblica

```
30 82 01 0a 02 82 01 01 00 da b2 12 ff fd 53
e0 e0 1e 12 db eb ba 9d 5b f6 be 56 14 37 f4
90 dc aa 52 4a 1f e1 4a 0b e7 1a cf f9 87 ac
b5 36 89 ed 4f 09 43 d9 4a d0 08 b4 2d 08
89 73 d8 b3 d4 49 0e 61 5d 4b 4e 45 b1 a5
23 0c 82 e1 ca 0f 98 78 6a e4 e4 5a 1d fd 88
32 ee cf fc 12 18 5d 3b 6f af b2 bf ec e9 8d c4
29 7c d8 57 e1 d8 15 d1 5d 18 73 ba d0 87 43
7d 8a 68 e5 55 c5 1d 07 ca 87 b1 65 c1 88 2b
85 50 ea fd f5 74 4d b6 48 a6 ef b0 5d a9 d6
b4 eb e3 76 e2 84 fa cb aa b4 9a 93 4a dd 99
4a 20 17 c7 86 df a5 a2 ae c0 2e a1 a7 3e 9a
73 7e 42 9a 1e bc 4b b7 b2 68 7b 24 40 7f 8c
08 14 3d d6 a4 4c fe ba 90 dc 78 95 3e 71 40
56 a1 21 9c de af d8 29 ea c0 e0 53 71 37 25
b4 8e a9 76 e3 59 db a8 39 ea ea 47 bb 20
b8 c7 cc fb 72 3b bd a8 59 c0 21 0b cf 5b b7
eb ad 96 b1 78 41 cd 24 f7 1c aa c9 02 03 01
00 01
```

UTILIZZO CHIAVE ESTESO

Critico	No
Scopo	Autenticazione server

INFORMAZIONI CHIAVE PUBBLICA

Algoritmo	Codifica RSA
Parametri	Nessuno
Dimensioni chiave pubblica	2048

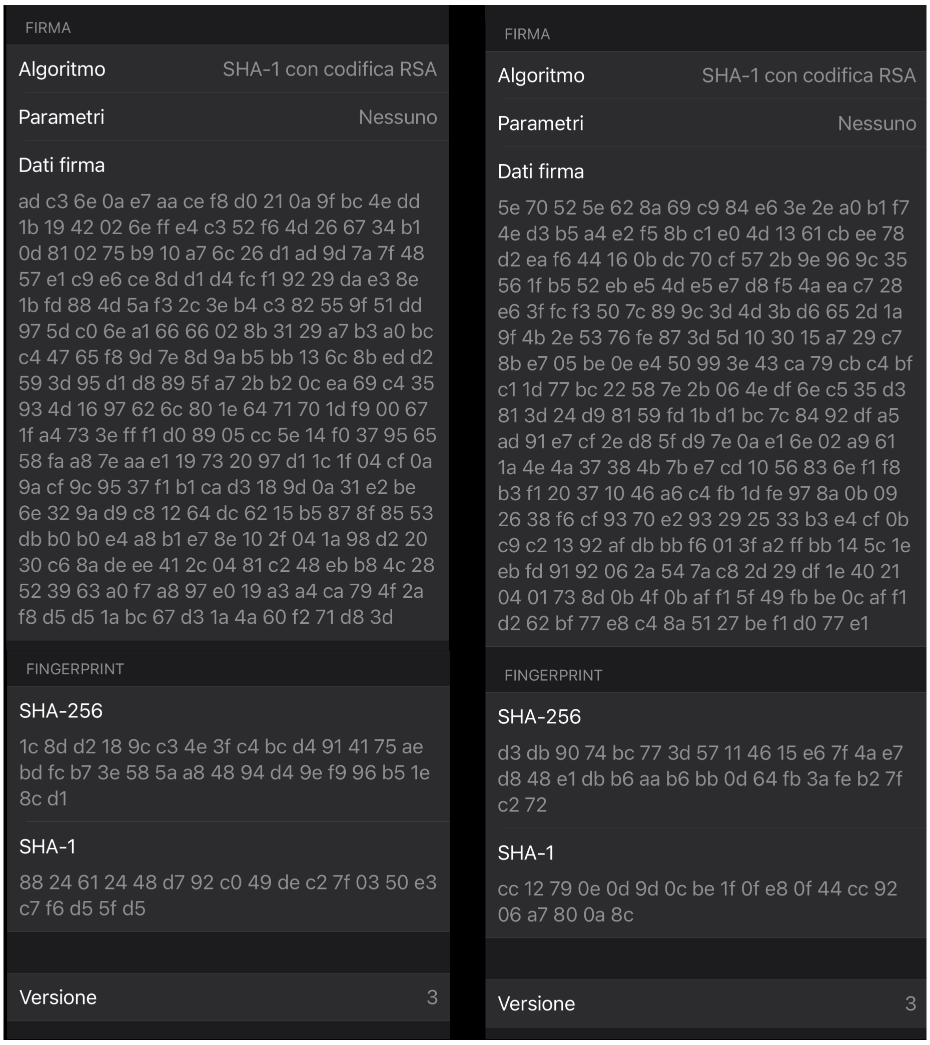
Dati chiave pubblica

```
30 82 01 0a 02 82 01 01 00 c2 8e 89 88 57 df
5e 72 43 66 1c 69 ff c6 54 46 70 60 fa c1 9f
e2 25 28 b5 83 71 83 30 51 07 86 9f db 50 9a
a2 39 61 3a 82 80 47 b0 8c 36 bd d4 18 dd
45 50 01 35 dc 10 12 6d 8c 3c cb 1a 8f 7a fb
5d c6 2f e9 9c 4a 5e 14 3f 6f 2a 54 5e 78 2d
dd ef e7 04 c7 c5 3f 36 bb 75 fa 36 68 ce d9
34 14 bc 83 af 47 cf a5 35 e9 67 88 c2 3b 41
82 57 95 a3 4c af 0a 60 db 2c 79 f2 f2 48 bf
31 51 03 7e eb 74 25 1b bc 7d aa 84 49 de d9
9b 56 15 2a 83 f8 6b 7a 68 c1 66 d5 4a 76 30
77 70 06 ba 7c 95 f4 4f 70 3b ff 73 bd 97 e9
64 db 42 25 5c b5 f4 b6 b1 72 ae 8d 56 dd 95
3e d8 3e 78 48 65 a7 2b 02 dd 9f 13 cf 06 ea
fb b9 8c c2 08 3c 72 a1 09 15 24 a1 bd 77 d0
6b 9b 62 88 35 4e db d8 c9 7f 41 52 be 44 4f
03 9a 01 a7 a1 5e d7 1c f6 0e 75 58 bd 95 cc
93 84 3a 0f d0 d4 8b f0 73 ed 15 02 03 01 00
01
```

UTILIZZO CHIAVE ESTESO

Critico	No
Scopo	Autenticazione server

Figure 15: On the left, public key contained in the certificate of the real RADIUS server. On the right, the public key of the rogue one.



FIRMA

Algoritmo SHA-1 con codifica RSA

Parametri Nessuno

Dati firma

```
ad c3 6e 0a e7 aa ce f8 d0 21 0a 9f bc 4e dd
1b 19 42 02 6e ff e4 c3 52 f6 4d 26 67 34 b1
0d 81 02 75 b9 10 a7 6c 26 d1 ad 9d 7a 7f 48
57 e1 c9 e6 ce 8d d1 d4 fc f1 92 29 da e3 8e
1b fd 88 4d 5a f3 2c 3e b4 c3 82 55 9f 51 dd
97 5d c0 6e a1 66 66 02 8b 31 29 a7 b3 a0 bc
c4 47 65 f8 9d 7e 8d 9a b5 bb 13 6c 8b ed d2
59 3d 95 d1 d8 89 5f a7 2b b2 0c ea 69 c4 35
93 4d 16 97 62 6c 80 1e 64 71 70 1d f9 00 67
1f a4 73 3e ff f1 d0 89 05 cc 5e 14 f0 37 95 65
58 fa a8 7e aa e1 19 73 20 97 d1 1c 1f 04 cf 0a
9a cf 9c 95 37 f1 b1 ca d3 18 9d 0a 31 e2 be
6e 32 9a d9 c8 12 64 dc 62 15 b5 87 8f 85 53
db b0 b0 e4 a8 b1 e7 8e 10 2f 04 1a 98 d2 20
30 c6 8a de ee 41 2c 04 81 c2 48 eb b8 4c 28
52 39 63 a0 f7 a8 97 e0 19 a3 a4 ca 79 4f 2a
f8 d5 d5 1a bc 67 d3 1a 4a 60 f2 71 d8 3d
```

FINGERPRINT

SHA-256

```
1c 8d d2 18 9c c3 4e 3f c4 bc d4 91 41 75 ae
bd fc b7 3e 58 5a a8 48 94 d4 9e f9 96 b5 1e
8c d1
```

SHA-1

```
88 24 61 24 48 d7 92 c0 49 de c2 7f 03 50 e3
c7 f6 d5 5f d5
```

Versione 3

FIRMA

Algoritmo SHA-1 con codifica RSA

Parametri Nessuno

Dati firma

```
5e 70 52 5e 62 8a 69 c9 84 e6 3e 2e a0 b1 f7
4e d3 b5 a4 e2 f5 8b c1 e0 4d 13 61 cb ee 78
d2 ea f6 44 16 0b dc 70 cf 57 2b 9e 96 9c 35
56 1f b5 52 eb e5 4d e5 e7 d8 f5 4a ea c7 28
e6 3f fc f3 50 7c 89 9c 3d 4d 3b d6 65 2d 1a
9f 4b 2e 53 76 fe 87 3d 5d 10 30 15 a7 29 c7
8b e7 05 be 0e e4 50 99 3e 43 ca 79 cb c4 bf
c1 1d 77 bc 22 58 7e 2b 06 4e df 6e c5 35 d3
81 3d 24 d9 81 59 fd 1b d1 bc 7c 84 92 df a5
ad 91 e7 cf 2e d8 5f d9 7e 0a e1 6e 02 a9 61
1a 4e 4a 37 38 4b 7b e7 cd 10 56 83 6e f1 f8
b3 f1 20 37 10 46 a6 c4 fb 1d fe 97 8a 0b 09
26 38 f6 cf 93 70 e2 93 29 25 33 b3 e4 cf 0b
c9 c2 13 92 af db bb f6 01 3f a2 ff bb 14 5c 1e
eb fd 91 92 06 2a 54 7a c8 2d 29 df 1e 40 21
04 01 73 8d 0b 4f 0b af f1 5f 49 fb be 0c af f1
d2 62 bf 77 e8 c4 8a 51 27 be f1 d0 77 e1
```

FINGERPRINT

SHA-256

```
d3 db 90 74 bc 77 3d 57 11 46 15 e6 7f 4a e7
d8 48 e1 db b6 aa b6 bb 0d 64 fb 3a fe b2 7f
c2 72
```

SHA-1

```
cc 12 79 0e 0d 9d 0c be 1f 0f e8 0f 44 cc 92
06 a7 80 0a 8c
```

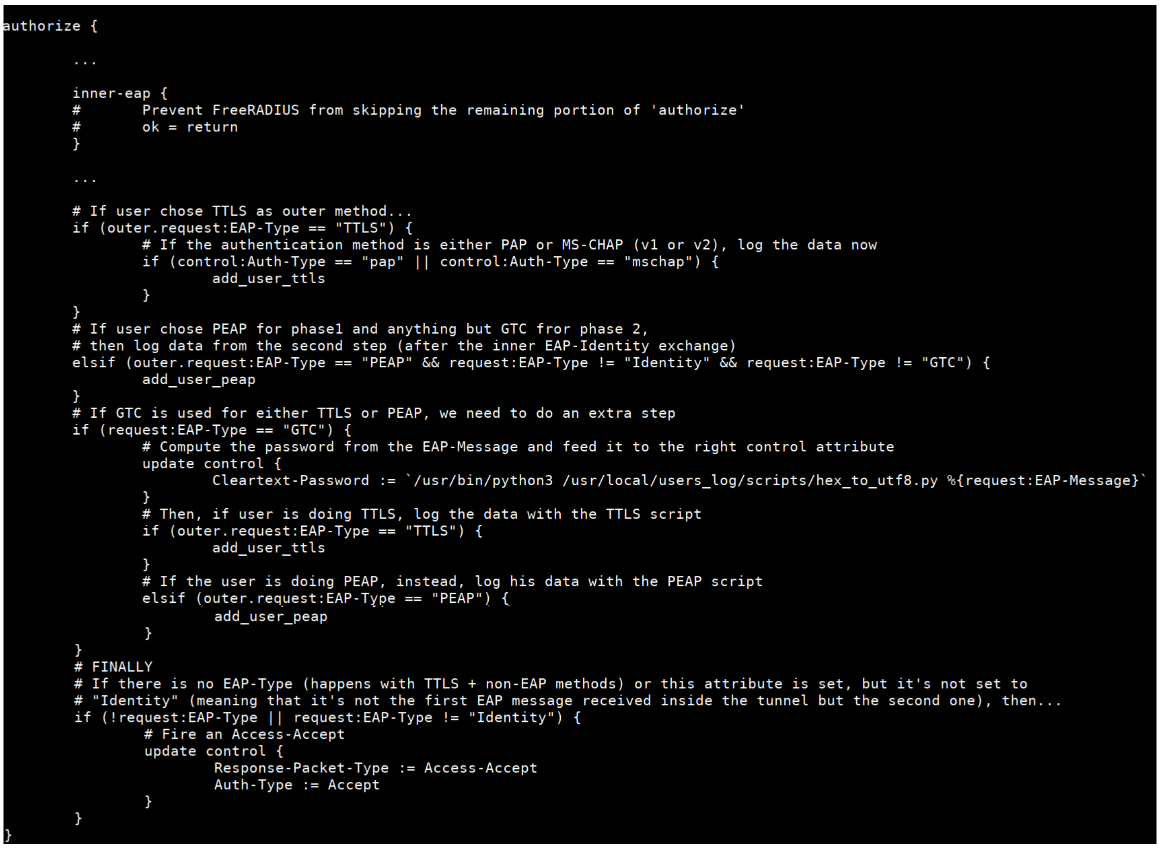
Versione 3

Figure 16: On the left, signature of the certificate of the real RADIUS server. On the right, the signature of the rogue one.

Again, discriminating between a real and a fake certificate by looking at the data contained inside them is quite hard, as it can be understood from the pictures above. For this reason, iOS users could be easily fooled into connecting to a rogue network.

15 Appendix C - Main modifications to FreeRADIUS

In this appendix, we will show and discuss the major modifications to FreeRADIUS that let us perform authentication of the users without knowing their passwords. As already said, we needed to alter the normal FreeRADIUS' operative flow, as otherwise every authentication attempt would have failed, since we don't know the password of any user. The modifications we made to achieve this goal mainly regard one specific file, called "inner-tunnel" in FreeRADIUS. In the following picture, the major modifications are shown.



```
authorize {

        ...

        inner-eap {
        #       Prevent FreeRADIUS from skipping the remaining portion of 'authorize'
        #       ok = return
        }

        ...

        # If user chose TTLS as outer method...
        if (outer.request:EAP-Type == "TTLS") {
                # If the authentication method is either PAP or MS-CHAP (v1 or v2), log the data now
                if (control:Auth-Type == "pap" || control:Auth-Type == "mschap") {
                        add_user_ttls
                }
        }
        # If user chose PEAP for phase1 and anything but GTC for phase 2,
        # then log data from the second step (after the inner EAP-Identity exchange)
        elsif (outer.request:EAP-Type == "PEAP" && request:EAP-Type != "Identity" && request:EAP-Type != "GTC") {
                add_user_peap
        }
        # If GTC is used for either TTLS or PEAP, we need to do an extra step
        if (request:EAP-Type == "GTC") {
                # Compute the password from the EAP-Message and feed it to the right control attribute
                update control {
                        Cleartext-Password := `/usr/bin/python3 /usr/local/users_log/scripts/hex_to_utf8.py %{request:EAP-Message}`
                }
                # Then, if user is doing TTLS, log the data with the TTLS script
                if (outer.request:EAP-Type == "TTLS") {
                        add_user_ttls
                }
                # If the user is doing PEAP, instead, log his data with the PEAP script
                elsif (outer.request:EAP-Type == "PEAP") {
                        add_user_peap
                }
        }
        # FINALLY
        # If there is no EAP-Type (happens with TTLS + non-EAP methods) or this attribute is set, but it's not set to
        # "Identity" (meaning that it's not the first EAP message received inside the tunnel but the second one), then...
        if (!request:EAP-Type || request:EAP-Type != "Identity") {
                # Fire an Access-Accept
                update control {
                        Response-Packet-Type := Access-Accept
                        Auth-Type := Accept
                }
        }
}
```

Figure 17: The most important modifications to the "inner-tunnel" file.

As shown in the picture above, the portion that needs to be modified to allow us to authenticate a user forcing an Access-Accept is the "authorize" section of the inner tunnel.

As it can be seen in the first few lines, the directive "ok = return" inside inner-eap needs to be commented out, otherwise the authentication server will instantly jump to the "authenticate" section once it receives a packet containing an EAP-Message attribute, leaving no room for us to do our own logic.

The remaining portion of code is structured to always log authentication data, that will be analyzed later. Since our AS proposes EAP-PEAP+EAP-GTC for phase 1 and phase 2 authentication, the authentication process can end up in one of three main branches:

- The user chose EAP-TTLS for phase 1 and any other method but EAP-GTC for phase 2. In this case, since we don't have time to negotiate the phase 2 method, we simply log data about the authentication and then fire an Access-Accept as soon as we receive the first message inside the TLS tunnel;

- The user chose EAP-PEAP for phase 1 and any other method but EAP-GTC for phase 2. In this case, our AS will propose EAP-GTC to the supplicant, but will receive an EAP-NAK plus the code of the EAP method desired by the user. Once the AS knows which protocol the supplicant wants to use, it fires an Access-Accept before starting that authentication protocol. Doing so ensures us that the supplicant will accept the Access-Accept packet, as we are not stopping an authentication protocol in the middle. Notice that we could fire the Access-Accept as soon as the supplicant sends the identity inside the TLS tunnel, but this would prevent us from knowing which phase 2 method was configured on by the user, an information we were really interested in;

- The user set the phase 2 method to "None" or "EAP-GTC". This means that, upon receiving the proposal by the AS, the supplicant will instantly start to use EAP-GTC, sending its password encapsulated inside an EAP message.

The last branch is surely the most interesting one as, by default, the EAP-GTC module would reject the user since the AS doesn't know the password. Without tweaks, if the supplicant agrees on doing EAP-GTC, the authentication run would fail inside the inner-eap module, as the AS doesn't know the right password. In order to compare the password stored in the database with the one sent by the supplicant, FreeRADIUS first looks for the user with that identity inside the database, then it loads the corresponding password in the "Cleartext-Password" attribute of the "control" list, which is a list of attributes used to control the flow of the authorization process. Once the password has been retrieved from the database, the EAP-GTC module will be started. Here, the two passwords (the one contained in control:Cleartext-Password and the one contained in the EAP message sent by the supplicant) will be compared. In order to make the EAP-GTC module succeed, we wrote a Python script that takes the EAP message sent by the supplicant, retrieves the password contained in it and returns it to FreeRADIUS, which will update the value of control:Cleartext-Password with the value returned by the script. Doing so ensures us that EAP-GTC will succeed, as control:Cleartext-Password surely contains the same value contained in the EAP message, as it was retrieved from that same message.

Extracting the clear text password from an EAP message is fairly easy once the format of an EAP message is known. To clarify the next steps, this picture shows the format of an EAP packet.

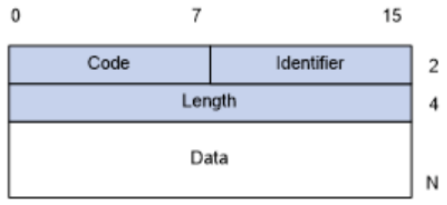


Figure 18: Format of an EAP packet.

To make an example, let's say that a supplicant sent the following EAP message, taken from a real-life authentication run we did on our evil-twin:

0x026f000a0668656c6c6f

This message can be decoded as follows:

1. 0x. It simply means that the message is in hexadecimal base;
2. 02. It means that this message is a response (requests have code 01, successes have code 03, failures have code 04, ...);
3. 6f. This is the identifier of the packet, which lets the supplicant and the AS identify this specific message among the ones sent during this authentication run;
4. 000a. This is the length, in bytes, of the whole EAP message. In this case, the length of the packet is 10 bytes. The message, as it can be seen, contains 20 hexadecimal characters (besides the initial 0x, which is not part of the packet), which are equivalent to 10 bytes;
5. 06. This is the numerical code assigned by IANA to EAP-GTC. Each EAP method has its own, unique code;
6. 68656c6c6f. This last portion contains the clear text password of the user. The password is UTF-8 encoded, so we simply need to convert the hexadecimal value to it binary representation and then read it according to UTF-8 encoding rules. In this case, the hexadecimal string translates to "hello" in UTF-8.

Looking at the whole message, in order to extract the password we need to discard the first 5 bytes, as the first four ones contain meta-data about the packet, while the fifth one always contains the identifier of the EAP method (06 for EAP-GTC). The remaining bytes of the packet all belong to the password, so they need to be decoded according to UTF-8 rules.

To summarize the scenario where a supplicant agrees to do EAP-GTC, we simply wait for it to send the EAP message containing the password, retrieve the password from the EAP packet (knowing which bytes to read it from), store it in the right FreeRADIUS' control variable and then normally launch the EAP-GTC module, which will now succeed for sure.

16 References

[1] S. Brenza, A. Pawlowski, and C. Pöpper. A practical investigation of identity theft vulnerabilities in eduroam. In *Proceedings of the 8th ACM Conference on Security and Privacy in Wireless and Mobile Networks (WiSec'15)*, 2015.

[2] T. Perković, A. Dagelić, M. Bugarić, and M. Cagalj. On wpa2-enterprise privacy in high education and science. *Security and Communication Networks*, 2020, Sept. 2020.

[3] B. Altinok. eduroam: Collect, track, hack. https://medium.com/@besimaltnok/eduroam-collect-track-hack-183e843f7efc, Last accessed on 2022-03-15.

[4] A. Bartoli, E. Medvet, and F. Onesti. Evil twins and wpa2 enterprise: A coming security disaster? *Computers and Security*, 74:1–11, 2018.

[5] A. Bartoli, E. Medvet, A. De Lorenzo, and F. Tarlao. (in) secure configuration practices of wpa2 enterprise supplicants. In *Proceedings of the 13th International Conference on Availability, Reliability and Security (ARES)*, 2018.

[6] V. Ramachandran. Cracking wpa/wpa2 personal and enterprise for fun and profit. In *Hacktivity Conference*, 2012.

[7] M. Ghering. Evil twin vulnerabilities in wi-fi networks. Bachelor's thesis, 2016.

[8] I. Palamà, A. Amici, F. Gringoli, G. Bianchi. "Careful with that Roam, Edu": experimental analysis of Eduroam credential stealing attacks. In *Proceedings of the 17th Wireless On-demand Network systems and Services Conference*, 2022.

[9] *WEP vs. WPA vs. WPA2 vs. WPA3: Wi-Fi Security Types Explained*, https://www.makeuseof.com/tag/wep-wpa-wpa2-wpa3-explained, last visited: 2022-01-04

[10] *Public key cryptography*, https://www.ibm.com/docs/en/ztpf/1.1.0.14?topic=concepts-public-key-cryptography, last visited: 2022-02-17

[11] *Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile*, https://datatracker.ietf.org/doc/html/rfc5280, last visited: 2022-01-04

[12] *Extensible Authentication Protocol (EAP)*, https://datatracker.ietf.org/doc/html/rfc3748, last visited: 2022-02-20

[13] *RFC 5931 - Extensible Authentication Protocol (EAP) Authentication Using Only a Password*, https://datatracker.ietf.org/doc/html/rfc5931, last visited: 2021-01-28

[14] *RFC 5216 - The EAP-TLS Authentication Protocol*, https://datatracker.ietf.org/doc/html/rfc5216, last visited: 2022-01-28

[15] *Extensible Authentication Protocol Tunneled Transport Layer Security Authenticated Protocol Version 0 (EAP-TTLSv0)*, https://datatracker.ietf.org/doc/html/rfc5281, last visited: 2021-02-16

[16] *[MS-PEAP]: Protected Extensible Authentication Protocol (PEAP)*, https://docs.microsoft.com/en-us/openspecs/windows_protocols/ms-peap/5308642b-90c9-4cc4-beec-fb367325c0f9, last visited: 2022-02-16

[17] *Overview of TLS-Protected EAP Methods*, https://www.interlinknetworks.com/app_notes/eap-peap.htm, last visited: 2022-02-18

[18] *Eduroam's official website*, https://eduroam.org/, last visited: 2022-02-19

[19] *Eduroam CAT's official website*, https://cat.eduroam.org/, last visited: 2022-02-19

[20] *PSA: Android 11 will no longer let you insecurely connect to enterprise WiFi networks*,
https://www.xda-developers.com/android-11-break-enterprise-wifi-connection/,
last visited: 2022-12-12

[21] *Extensible Authentication Protocol (EAP) Registry*,
https://www.iana.org/assignments/eap-numbers/eap-numbers.xhtml, last visited: 2022-02-18