Breaking PPTP VPNs via RADIUS Encryption

Matthias Horst, Martin Grothe, Tibor Jager, and Jörg Schwenk

Horst Görtz Institute, Ruhr-University Bochum
{matthias.horst, martin.grothe, tibor.jager,
joerg.schwenk}@rub.de

Abstract. We describe an efficient *cross-protocol* attack, which enables an attacker to learn the VPN session key shared between a victim client and a VPN endpoint. The attack recovers the key which is used to encrypt and authenticate VPN traffic. It leverages a weakness of the RADIUS protocol executed between a VPN endpoint and a RADIUS server, and allows an "insider" attacker to read the VPN traffic of other users or to escalate its own privileges with significantly smaller effort than previously known attacks on MS-CHAPv2.

1 Introduction

The *Point-to-Point Tunneling* (PPTP) protocol [5] implements a confidential and authenticated virtual private network (VPN) tunnel in public computer networks like the Internet. In this work, we analyze the security of PPTP using MS-CHAPv2 in combination with a RADIUS authentication server. This is a standard setting, which is used in large-scale and enterprise networks, where RADIUS is used to centralize user management and to perform authentication for different applications. Large scale analysis of public VPN service providers shows that over 60% of these still offer PPTP [14].

Contributions. We describe an efficient *cross-protocol* attack, which enables an attacker to learn the VPN session key shared between a victim client and a VPN endpoint. The attack recovers the key which is used to encrypt and authenticate VPN traffic, usually with the Microsoft Point-to-Point Encryption (MPPE) [9] scheme. The attack leverages a weakness of the RADIUS protocol executed between the VPN endpoint and the RADIUS server.

VPN session establishment with RADIUS authentication. In order to be able to sketch our attack, we first describe how a VPN session is established with RADIUS authentication. VPN session establishment with RADIUS involves three parties:

- The *client* which connects to the VPN endpoint. It shares a secret password with the RADIUS server. The RADIUS server is used to authenticate the client (or the user that uses this client). We assume that this password is a strong, high-entropy password, such that a dictionary attack is infeasible.
- The *VPN endpoint* relies on the RADIUS server to authenticate users. It shares a *RADIUS secret* S with the RADIUS server. We assume that S is a cryptographically strong high-entropy key.

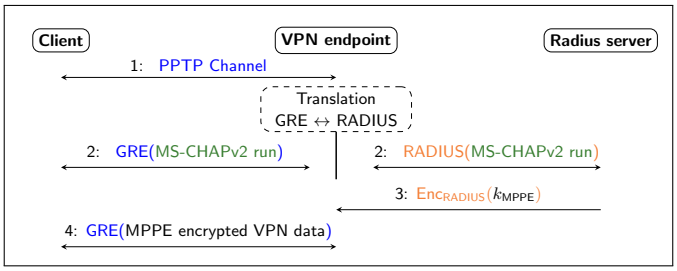

Fig. 1. Protocol Overview

– The *RADIUS server* is a trusted party, which performs user authentication on behalf of the VPN endpoint, using that it shares the password with the client and the RADIUS secret S with the VPN endpoint.

Establishment of a VPN session works as follows (cf. Figure 1 for an overview, detailed version Figure 4).

1. The client initiates a PPTP session with the VPN endpoint.
2. At the beginning of the PPTP session, it authenticates itself by running the MS-CHAPv2 protocol. The VPN endpoint relays all MS-CHAPv2 messages between the client and the RADIUS server. As a result of the MS-CHAPv2 protocol, client and RADIUS server obtain a shared session key k_{MPPE} for the connection between client and VPN endpoint. Additionally, the VPN endpoint transmits a random nonce, called the *Request Authenticator* (Req_{Auth}) to the RADIUS server.
3. The RADIUS server uses the RADIUS secret S shared with the VPN endpoint to encrypt and send k_{MPPE} to the VPN endpoint. Here the so-called *RADIUS encryption* scheme Enc_{RADIUS} is used, which *essentially* computes a ciphertext $Enc_{\mathsf{RADIUS}}(k_{\mathsf{MPPE}})$ encrypting k_{MPPE} as
$$Enc_{\mathsf{RADIUS}}(k_{\mathsf{MPPE}}) = (\mathsf{Salt}, \mathrm{MD5}(S||Req_{\mathsf{Auth}}||\mathsf{Salt}) \oplus k_{\mathsf{MPPE}})$$
where Salt is a short random 11 bit Salt and Req_{Auth} is the random nonce selected in Step 2 by the VPN endpoint. (a full description of RADIUS encryption can be found in subsection 2.3)
4. The VPN endpoint decrypts this message. Now the client and the VPN endpoint share a session key k_{MPPE}, which can be used to encrypt VPN payload data, using the Microsoft Point-to-Point Encryption (MPPE) protocol.

High-level attack description. Our attack is based on the following observations about RADIUS encryption as used in the setting described above.

– The "pseudorandom" value $\mathrm{MD5}(S||Req_{\mathsf{Auth}}||\mathsf{Salt})$ used to encrypt k_{MPPE} depends *deterministically* on S, Req_{Auth}, and Salt.
– The *same* value of S is used to encrypt *all* ciphertexts sent from the RADIUS server to the VPN endpoint.
– Salt has only 11 bits of entropy, therefore it is very likely that it is repeated in different ciphertexts sent from the RADIUS server to the VPN endpoint.

– The Req_{Auth} is a random nonce with high entropy (128 bits), however, it is *chosen by the VPN endpoint* and transmitted in *plain and unauthenticated* form from the VPN endpoint to the server.

Our attack leverages these observations as follows. We consider a setting with an attacker that meets the following two requirements:

– The attacker is able to monitor all data exchanged between the VPN endpoint and the RADIUS server, and it is able to inject packets.
– The attacker is an "insider", who is also able to establish VPN connections (but possibly with lower permissions than other users), whose goal is to learn the session key of *another* user.

This is a very practical setting in many applications of PPTP. We require the attacker to perform only a very small amount of computations, which could even be performed on a constrained device or a smartphone within a very short time (a few seconds). We also sketch below how the assumption of an "insider" attacker can be removed.

1. While the victim initiates a VPN connection, the attacker observes all messages exchanged between VPN endpoint and RADIUS server. In particular, it records Req_{Auth} and $Enc_{\mathsf{RADIUS}}(k_{\mathsf{MPPE}}) = (\mathsf{Salt}, \mathrm{MD5}(S||Req_{\mathsf{Auth}}||\mathsf{Salt}) \oplus k_{\mathsf{MPPE}})$.
2. The attacker also initiates a VPN session *as an honest user*[1] and proceeds as follows:
   (a) The attacker runs the MS-CHAPv2 protocol to establish a shared session key k^*_{MPPE} shared with the RADIUS server.
   (b) When the VPN endpoint sends a random RADIUS Request authenticator Req^*_{Auth} to the RADIUS server, then the attacker *replaces* Req^*_{Auth} with the previously recorded value Req_{Auth} sent from the VPN endpoint.
   (c) The RADIUS server will respond to the VPN endpoint with a RADIUS encryption

$$Enc_{\mathsf{RADIUS}}(k^*_{\mathsf{MPPE}}) = (\mathsf{Salt}^*, \mathrm{MD5}(S||Req_{\mathsf{Auth}}||\mathsf{Salt}^*) \oplus k^*_{\mathsf{MPPE}})$$

   If $\mathsf{Salt}^* = \mathsf{Salt}$ (which happens with high probability, because the salt is a short random string of only 11 bits), then the attacker is able to use the fact that it knows k^*_{MPPE} to easily compute

$$\mathrm{MD5}(S||Req^*_{\mathsf{Auth}}||\mathsf{Salt}^*) = \mathrm{MD5}(S||Req_{\mathsf{Auth}}||\mathsf{Salt})$$

   from $Enc_{\mathsf{RADIUS}}(k^*_{\mathsf{MPPE}})$. This is sufficient to decrypt the session key contained in the message $Enc_{\mathsf{RADIUS}}(k_{\mathsf{MPPE}})$ of the victim's session.

Experimental analysis of the attack. We have implemented the attack in Python on a Ubuntu Linux machine. The target RADIUS server was the FreeRadius Server 3.0.10.

Our analysis shows that computing the session key of a victim user takes about 62 seconds in our setting on average.

[1] Recall here that in the basic setting we assume that the attacker is an "insider", which aims at learning the key k_{MPPE} of the victim in order to read the traffic or to escalate its own privileges.

Comparison to other attacks on MS-CHAPv2. It is well-known that MS-CHAPv2 is cryptographically weak, as it is based on the DES encryption scheme with 56 bit keys [13]. The previously best known attacks on MS-CHAPv2 were passive (=eavesdropping) attacks that recover the DES key, which required an exhaustive search over the key space of size 2^{56} (which is feasible on high-performance hardware, but relatively expensive) or were based on the use of low-entropy passwords [13,7].

In contrast, we show an active attack allowing to break MS-CHAPv2 authentication in PPTP with RADIUS authentication with *significantly* smaller effort of only 2^{14}, which is feasible even without access to high-performance hardware.

Extension to "outsider" attackers. Our attack assumes an "insider" attacker, but we note that it generalizes easily to "outsider" attackers as well. An outsider would first run the attack from Schneier and Mudge [13] to break MS-CHAPv2, in order to recover the secret of one user to become an "insider", and then mount our attack.

The main advantage of this approach is that the attacker has to execute the (feasible, but relatively expensive) attack of Schneier and Mudge only *once*, while without our attack technique he would have to executed it once for each victim user.

Further related work. Schneier and Mudge [13] as well as Eisinger [4] analyzed the security of MS-CHAPv2 and showed the maximum security is one full DES key space search. MPPE security was analyzed most recently by Patterson et. al [10], who exploited biases in the RC4 keystream, in order to mount plaintext recovery attacks. Downgrade attacks on PPTP were showed by Ornaghi et. al [8], which tried to force PAP or MS-CHAPv1 as authentication protocol instead of MS-CHAPv2.

2 Foundations

Our attack utilizes three different protocols, as shown in Figure 1 and therefore can be split into three different parts:

1. The first part is the setup of an PPTP channel over any PPP channel between the client and the VPN endpoint using the Link Control Protocol and Network Control Protocol. All data is transfered encapsulated in GRE packets. This is described in subsection 2.1.
2. The second part is the login procedure of the client at the RADIUS server. Here the data is transported again with GRE on the side between client and VPN endpoint and is then repacked into RADIUS packets and send from the VPN endpoint to the RADIUS server. This part is decribed in subsection 2.2
3. After the client is successfully logged in, he and the RADIUS server both compute a key k_{MPPE} and the RADIUS server encrypts this key using the RADIUS encryption subsection 2.3 and sends it to the VPN endpoint. Now that the client and VPN endpoint both have the same key, they derive a session key from that and can start MPPE encrypted data communication. This part is described in subsection 2.4.

After we describe this setting in detail, we introduce an attacker that can get the key k_{MPPE}, with a few messages send to the RADIUS server under certain assumptions. We will then show, how the attacker can use the key to decrypt all messages from the secured MPPE channel. Our attack is described in section 3.

2.1 PPTP

The Point-to-Point-Tunneling Protocol (PPTP) was designed to allow for clients that are not part of a network to tunnel their data trough a Point-to-Point protocol to that network to extend the original one with a virtual one. This allows to create Virtual Private Networks (VPNs). The Point-to-Point method was chosen, so that it was not necessary to have a working Ethernet connection between the networks, but phone communication or others could also be used.

PPTP uses a control channel over TCP and a second channel that is encapsulated in GRE to transfer the data.

PPP The Point-to-Point-Protocol (PPP) was introduced in 1994 in RFC 1661 [1]. It is a layer-2 protocol to transmit arbitrary data packets over a full duplex point-to-point connection that can be established over many underlying systems.

Aside from the channel that transmits the actual data, the PPP uses two distinct protocols to agree how the channel is build:

1. Link Control Protocol (LCP)
2. Network Control Protocol (NCP)

The LCP focuses on all management between the two parties constructing the channel, while the NCP controls how the selected payload protocol is used in the transfer later.

While the original PPP protocol was designed to allow transfer over Point-to-Point connections, the protocol was extended with Point-to-Point-over-Ethernet to also work in a Ethernet environment that is not a direct two point connection. This allows the use of all protocols based on PPP to be used over the Internet. This allows PPTP Endpoint to work directly with clients coming over the Internet and others using phone-lines with the same protocol.

PPTP The PPP protocol itself does not offer security protection. As a result the Point-to-Point-Tunneling-Protocol (PPTP) was designed. In the beginning when PPP was only used over direct Point-to-Point connected endpoints, the security was derived from this direct connection. Now that it also possible to use the Internet as the underlying layer, these security guarantees are not valid anymore and additional security is needed. In the original PPTP specification in RFC 2637 [5], which was mainly driven by Microsoft, Ascend and a few others, were already different authentication mechanism introduced. The PPTP supports PAP, CHAP and the Microsoft version MS-CHAP. After MS-CHAP v1 was proven to be insecure, Microsoft developed v2. This standard has some security difficulties, but until today still has a complexity of 2^{56} for an attacker to get the password of a client. We will make use of this attack later. Besides the authentication Microsoft introduced the encryption Microsoft Point-to-Point Encryption (MPPE), which we describe in subsection 2.4.

Microsoft used PPTP as default way to construct secure VPN connections in its operating system Windows for a long time. Today still all Windows OSs have build in support for PPTP with the Microsoft authentication mechanisms MS-CHAPv1/v2. Further 60% of publicly available VPN service providers still offer PPTP as a possible

VPN mechanism [14]. Also every smartphone with Android or iOS supports PPTP by default.

GRE The Generic Routing Encapsulation (GRE) is an older standard described in RFC 1701 [6], mainly developed by Cisco, that allows PPTP to transfer its data over many different PPP protocols. Based on the fact that it is a data format, it does not influence the security and will not be discussed in more detail.

2.2 MS-CHAPv2

During the setup of a PPTP connection a variety of protocols can be utilized to authenticate the users. MS-CHAPv2 is one example. It is used together with Microsoft's implementation of the Point-to-Point tunneling Protocol. In Microsoft environments PPTP is used together with Microsoft Point-to-Point Encryption algorithm (MPPE). The MS-CHAPv2 protocol is depicted in Figure 2. An example protocol run is as follows. First the client requests an authenticator challenge from the server. The server then creates a 16 byte random authenticator challenge (C_S) and sends it back to the requesting client.

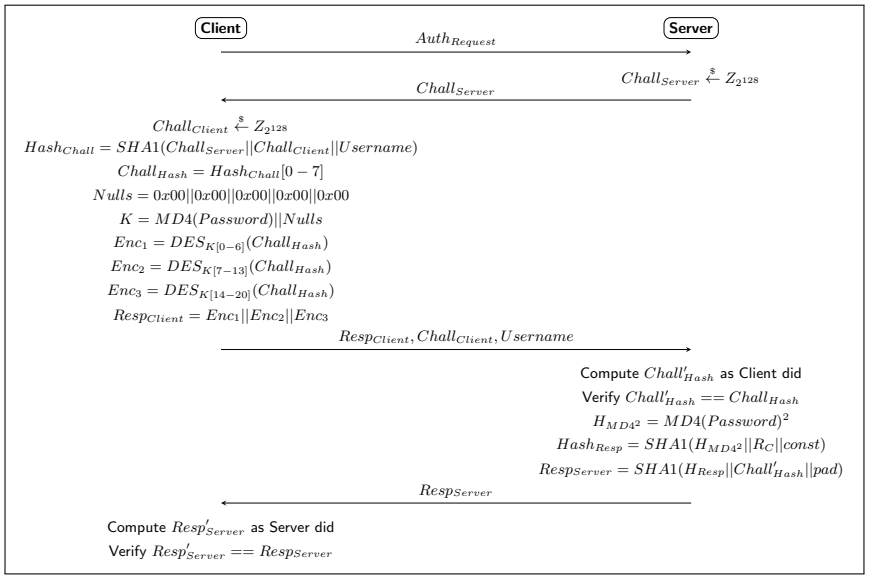

Fig. 2. MS-CHAPv2 Protocol run

Next the client creates a new random 16 byte long challenge. This challenge together with the the *name of the client user* and the challenge created on the server side are hashed via SHA-1 and the first 8 bytes results in the client hash ($\mathsf{Chall}_{\mathsf{Hash}}$). For the creation of the client response of the server challenge a key for the DES algorithm is created by hashing the user's password via the message digest algorithm 4 (MD4). The

resulting hash is afterwards concatenated with 5 zero bytes. These constant 5 bytes lead to some major security issues as described by Schneier et. al [13]. The resulting bytes stored in k are then split up into 3 keys (k_1, k_2, k_3) and used for 3 different DES encryptions with $\mathsf{Chall}_{\mathsf{Hash}}$ as input data for every encryption. All ciphertexts are concatenated and stored as the client response in R_{C}. The values R_{C}, C_{C} and U_{Name} are sent back to the server. To verify the credentials of the user, the server recreates the client response. Therefore, it uses the password it stored for the corresponding user name received with the client response, as well as the $\mathsf{Chall}_{\mathsf{Hash}}$ recomputed by the server. It then compares its created client response with the received R_{C}. In case the values are equal the server continues with the authentication process. The next protocol message created by the server is the server response R_{S}. As a preparation the server double hashes the password of the user with the MD4 algorithm. The resulting hash value is used together with client response and a constant string value (const) [2]. This hash value ($\mathcal{H}_{\mathsf{SHA1}}$) is used together with the previous created $\mathsf{Chall}_{\mathsf{Hash}}$ and the constant string value pad [3] as input for another SHA1 run. The result is named server response R_{S} and send to the client, which verifies the value. In case the verification ends successfully the mutual authentication process is also completed successfully. [13,15]

Schneier et. al Attack on MS-CHAPv2 As mentioned earlier, MS-CHAPv2 can be broken by doing just one exhaustive key search of the DES key space. This is possible due to the fact that the input data ($\mathsf{Chall}_{\mathsf{Hash}}$) for all three DES encryption runs stays the same (cf. Figure 2). Thus, 2^{56} encryption executions are necessary to find all three keys ($K[0 \ldots 6]$, $K[7 \ldots 13]$ and $K[14 \ldots 20]$). This is accomplished by trying every $K \in Z_{2^{56}}$ as encryption key, when $\mathsf{Chall}_{\mathsf{Hash}}$ is input into the DES encryption and compare the corresponding result with Enc_1, Enc_2 and Enc_3. As soon as one key matches, it can be stored and the search continues until all three keys are found [13].

2.3 RADIUS Encryption

The RADIUS protocol defines its own encryption scheme. This scheme is mandatory for PPTP, and is used by default in software like the FreeRADIUS [11] server. The algorithm is defined in two RFCs: RFC 2865 [12], which is the default RFC for RADIUS, defines how RADIUS encrypted user passwords are send. RFC 2868 [17] defines how RADIUS is used in tunnel scenarios. Both versions only differ in the point that the RFC for the tunnel scenarios adds an additional Salt. We will focus on the tunnel version from now on, because this version is used to encrypt the MPPE key (k_{MPPE}).

The specification does not take care of key management for the RADIUS encryption. As a result the shared key has to be established manually on the VPN endpoint and the RADIUS server. If this shared key is of low-entropy, it can be computed using dictionary attacks [2]. Therefore, we assume that only high-entropy keying material is used.

Basically the RADIUS encryption is a stream cipher, with an input seed consisting of the (static) RADIUS secret and some (pseudo-)random values (RA and Salt for the

[2] "Magic server to client constant"

[3] "Pad to make it do more than one iteration"

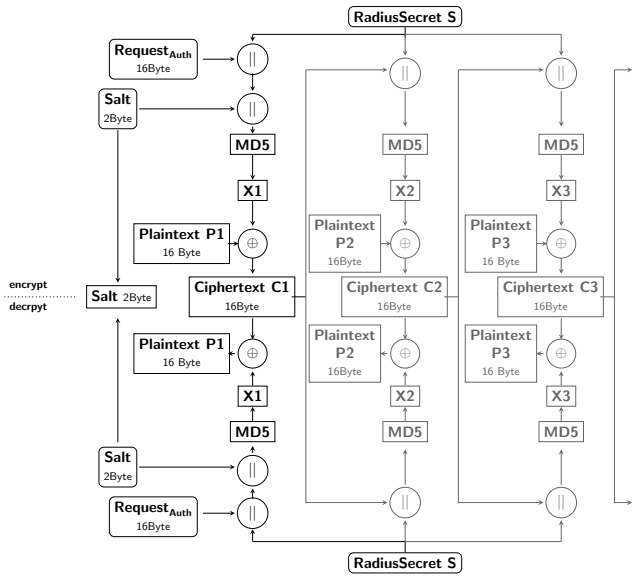

Fig. 3. RADIUS encryption and decryption

first block, c_i and Salt for the others) hashed using a MD5 hash function to generate the keystream. The ciphertext is then generated by computing the XOR of the keystream and the plaintext (cf. Figure 3).

The pseudorandom values are used to prevent the keystream output of MD5 from always being identical for a single RADIUS secret. The Request Authenticator (Req_{Auth}) is chosen by the client before encrypting the data and is 16 bytes long.

The Salt is 2 bytes long. The first bit is fixed to 1, indicating that the salt was chosen by the server. This is followed by a 4 bit offset that is always incremented by 1. The remaining 11 bits are chosen at random. The salt is transmitted as a prefix to the encrypted data.

In the formal way the RADIUS encryption can be defined as two algorithms (Enc, Dec) with

$$c = \mathsf{Enc}(S, Req_{Auth}, p, SaO) \quad \text{and} \quad p = \mathsf{Dec}(S, Sa, Req_{Auth}, c)$$

Request Authenticator In the RADIUS standards, there is some confusion on the term "Request Authenticator" (Req_{Auth}). In the original version of RADIUS encryption [12], the client selects Req_{Auth} and uses it as nonce and an IV for the encryption. The IV is used together with the RADIUS secret to encrypt data which should be sent to the server. The Req_{Auth} is a 16 byte value and stored in the message field named RADIUS Authenticator (RA). In case no encryption is used, Req_{Auth} is just a nonce value and not an IV in the RADIUS encryption. Thus, every RADIUS message from the client to the server contains a new value stored in the field RA.

Of course there are also RADIUS messages send from the server to the client. The server use the message field RA for authentication purposes. It creates a Message Authentication Code (MAC) called "Response Authenticator". The MAC is computed over

Algorithm 1 Algorithm Encrypt

Input: Key Material:
RadiusSecret S $\in \{ASCII\}^{sc}$ with $1 < sc \leq 16$
$Req_{\mathsf{Auth}} \in \{0,1\}^{128}$
Last SaltOffset SaO
Plaintext p $\in \{0,1\}^{t}$ as
$[p_1 \in \{0,1\}^{256}, p_2, ...p_n \in \{0,1\}^{256}]$ with $n = \lceil \frac{t}{256} \rceil$
Output: Ciphertext C $\in \{0,1\}^{t}$ as
$[c_1 \in \{0,1\}^{256}, c_2, ...c_n \in \{0,1\}^{256}]$

1: $SaO = (SaO + 1)\ mod\ 16$
2: $SaR \xleftarrow{\$} \{0,1\}^{15}$
3: $Sa = 1 \| SaO \| SaR$
4: $r = MD5(S \| Req_{\mathsf{Auth}} \| Sa)$
5: **for** $i = 1; i < n; i++$ **do**
6: $\quad c_i = p_i \oplus r$
7: $\quad r = MD5(S \| c_i)$
8: **end for**
9: **return** Sa, c as $[c_1, c_2, ..., c_n]$

Algorithm 2 Algorithm Decrypt

Input: Key Material:
RadiusSecret S $\in \{ASCII\}^{sc}$ with $1 < sc \leq 16$
Salt Sa with
SaltOffset $0 < SaO < 15$
SaltRandom SaR $\in \{0,1\}^{15}$
$Req_{\mathsf{Auth}} \in \{0,1\}^{128}$
Ciphertext c $\in \{0,1\}^{t}$ as
$[c_1 \in \{0,1\}^{256}, c_2, ...c_n \in \{0,1\}^{256}]$ with $n = \lceil \frac{t}{256} \rceil$
Output: Plaintext p $\in \{0,1\}^{t}$ as
$[p_1 \in \{0,1\}^{256}, p_2, ...p_n \in \{0,1\}^{256}]$

1: $r = MD5(S \| Req_{\mathsf{Auth}} \| Sa)$
2: **for** $i = 1; i < n; i++$ **do**
3: $\quad p_i = c_i \oplus r$
4: $\quad r = MD5(S \| c_i)$
5: **end for**
6: **return** p as $[p_1, p_2, ..., p_n]$

are all encrypted message fields, except the RA field. Afterwards, the MAC ($Resp_{Auth}$) is stored in this Req_{Auth} field. As a consequence the field cannot be used to store the new IV needed for the used RADIUS encryption. Thus, the IV (Req_{Auth}) from the last message from the client to the server is used. In short, the client controls which IV is used by the server for the encryption.

The RADIUS RFC [12] defines that the RAs should not be used twice with the same RADIUS secret, but this is not checked by the server. This behavior allows a successful attack against PPTP.

2.4 MPPE

For the PPTP protocol an additional encryption protocol is needed in order to encrypt the user data. Because PPTP itself does not offer such an encryption, Microsoft presented the Microsoft Point-to-Point Encryption protocol (MPPE) [9]. MPPE offers 3 different lengths for the key: 40, 56 and 128 bits. We assume that the strongest option with 128 bits is used, because the other options were designed to fulfill other regulations like export limitations.

MPPE was designed for the use case of PPTP. As a result the protocol expects an open PPTP channel and that a key was derived there. The encryption is done by using the standard RC4 algorithm. MPPE only defines how the keys and the data are fitted to be used in the RC4 encryption algorithm.

MPPE offers the functionality to exchange the key while transmitting data and to synchronize keys again if the synchronization is lost at some point. This allows MPPE to change the keys after a set schedule. The keys only depend on older keys. If the first key is compromised, all others could be computed by an attacker. Thus, there is no real key freshness. In this paper we will focus only on the first key for a session.

Key Derivation MPPE utilizes the keys derived by other protocols for its own key derivation. This is then used in the RC4 algorithm. The protocols from which MPPE can derive keys are MS-CHAPv1, MS-CHAPv2 and TLS as specified in RFC 3079 [16].

For MS-CHAPv2 it works as follows: the key is split in two halves, one for sending data from the client to the VPN endpoint called "Send-Key" and one for the other way around called "Recv-Key". This is done by hashing the MS-CHAP key together with different magic constants. The Send-Key is computed as follows:

$$\mathsf{Send\text{-}Key} = \mathcal{H}_{\mathsf{SHA1}}(Key \| pad \| const\_send \| pad)$$

and the Recv-Key the same way with another constant. These keys are used as the starting point for the session keys that are used for the encryption. The actual keys used for the encryption are called session keys, and the first is derived only from the Send-Key and Recv-Key, while the following also use the last session key. Here again as hashing function only SHA1 is used. The key is derived as follows:

$$\mathsf{Session\text{-}Send\text{-}Key} = \mathcal{H}_{\mathsf{SHA1}}(\mathsf{Send\text{-}Key} \| Pad \| \mathsf{Send\text{-}Key} \| Pad)$$

In consecutive runs the second Send-Key in the formula would be exchanged with the last Session-Send-Key. When the keys are switched is depending on the configuration of the VPN endpoint. It allows for sessions that run over a long period of time to change the keys in between, without having to do a full restart.

Format of Key Fields in RADIUS. Microsoft extended the RADIUS format with vendor specific attributes in order to be able to transport keys in encrypted form. This was necessary, because no fields for these purposes were available before. Thus, Microsoft introduced the field *MS-MPPE-RECV-KEY* and *MS-MPPE-SEND-KEY* to transmit the keys in both directions for MPPE.

Today these fields are reused to transport keys for other protocols that do not have any connection to the MPPE protocol. One of the best known protocol that use these field are the EAP protocols like EAP-(T)TLS that use this field to transmit the PMK.

3 Attack

For our attack we will first describe the scenario and its requirements. Further we will show that this scenario is quite common and can even be relaxed if our attack is mixed with other attacks later. Then we will introduce our known-plaintext attack against RADIUS that allows us to learn about the key material that is used for a key stream of a stream cipher. Then we will show that this attack which is not specific for the PPTP VPN scenario can be used to mount an chosen-ciphertext attack against the VPN which drastically reduces the complexity of known attacks against it. Finally we show our real life test results of this attack against the well used RADIUS implementation FreeRADIUS.

3.1 Scenario

Our attack works in every scenario, in which the attacker can wiretap and inject packets to the local network of the VPN endpoint and RADIUS server. An example of such a scenario is a university. In general students have access to the network via wireless

LAN. Due to the high amount of consumer devices (e.g. smartphones, tablets, notebooks) there is a demand for wireless access, therefore universities have many access points installed. In general these access points are not protected well physically and can simply be exchanged against another arbitrary device by an attacker. This way a MitM attack is easy to mount. Many universities also use RADIUS to manage the authentication process of the users. In addition, universities often provide PPTP VPN services so students can access university servers from abroad (e.g. ERASMUS), Internet cafes or from home. A further requirement are valid MS-CHAPv2 credentials for the VPN.

So we will, from this point on say that the attacker fulfills the following requirements:

1. The attacker can act as a MitM between an VPN device and the RADIUS Server
2. The network offers PPTP with MS-CHAPv2 (also other internal protocols like MS-CHAPv1 would also work, but they are already broken)
3. The attacker has valid credentials for the network

Getting valid MS-CHAPv2 credentials is as easy as applying for an arbitrary study in the university scenario. But this assumption can also easily be fulfilled for other company networks, for which one could not just register. Here the attack presented by Schneier et. al [13], described in section 2.2, can be executed to brute-force the credentials of an arbitrary user from any wiretapped PPTP connection. Note, that breaking the weakest credentials of some user is enough for our attack. Afterwards, these credentials can be used to run our attack against every other user using the PPTP VPN of the closed network.

3.2 Known-Plaintext Attack on RADIUS Encryption

In this section we will introduce our attack to get the key material that is sent encrypted with the RADIUS encryption. We will use a known plaintext attack on the RADIUS encryption that allows us to recover the first 16 bytes of the key material of a target ciphertext. We have already shown that PPTP only allows keys with the length of 40, 56 or 128 bits, so this always gets the attacker the complete key in the final attack in the next chapter.

Overview Known-Plaintext Attack on RADIUS to partially decrypt MPPE. The problem of the RADIUS encryption is that the Req_{Auth} is used as an initialization vector IV for the encryption but is not chosen by the RADIUS server who performs the encryption, but by the client. This allows for a simple and fast attack on the RADIUS encryption that has the following features:

1. To decrypt a value that is encrypted with the RADIUS encryption one only needs the output $X1$ of the MD5 hash function (cf. Figure 3). This value is the key stream that is used to be xored with the plaintext.
2. This MD5 result depends on the 3 inputs: RADIUS secret, the Req_{Auth} and the Salt. While the secret only changes if manually reconfigured, it can be assumed to be constant. To prevent an easy known-plaintext attack a Req_{Auth} is chosen with a length of 128 bits, so that it is statistically unlikely that one RA is chosen twice in a measurable amount of time.

3. The Salt is only part of the encryption if the tunneled RADIUS mode is used. This is done to add an easy way to check for the correct order by adding the 4 bit salt counter. In addition, the salt differs in the first bit if send from the client to the server or the other direction, so that a message encrypted in one direction cannot be injected in traffic in the opposite direction.

The attack is using the fact that the Req_{Auth} was designed to be chosen by the party, which use the RADIUS encryption to encrypt messages. This is not true in case the message originates from the server. Then the field RA is used in another way namely to store the Response Authenticator. As described in section section 2.3, the server utilizes the last received Req_{Auth} as IV for the encryption instead of choosing it on its own. This is the point the attack applies. In RADIUS the messages from the client to the server are not integrity protected (just from server to client), so that an attacker could easily exchange this Req_{Auth} and force two of the three input values for the MD5 to be the same.

Now only the Salt remains changing, which based on its structure, is not a big problem. The first bit is always 1 if the message is from the server and the next 4 bits are a counter that increases by 1 per use in the encryption. But the tunneled encryption is used twice in the RADIUS message, so that it is the same every 8 protocol runs. Only the remaining 11 bits are random meaning an attacker can get every 8 protocol runs a chance with probability of 2^{-11} to get the same key stream. This allows for an efficient known-plaintext attack against the tunneled version of the RADIUS encryption. In principal, the attack works also for the non-tunneled version from RFC 2865 [12], but here in practice the Req_{Auth} is always chosen by the sender and not the responding party.

3.3 Chosen-Ciphertext Attack on PPTP

Chosen RA Attack on RADIUS Encryption The attack to get the key-stream for MPPE k_{MPPE} uses the known-plaintext attack idea outlined in subsection 3.2 to get the Req_{Auth} and combines it with the PPTP scenario.

As described previously the client and VPN endpoint need a shared key to establish a connection secured by MPPE. But only the client and RADIUS server share any common keys. The VPN endpoint and the RADIUS server share the RADIUS secret S. So the idea is for the VPN endpoint to just relay the MS-CHAPv2 login messages from the client to the RADIUS server and if the login is successful, to get key material from the RADIUS server. The client can compute the same key material from the MS-CHAPv2 protocol results (cf. subsection 2.2).

The key material k_{MPPE} is transferred in the RADIUS packet from the RADIUS server to the VPN endpoint. It is transferred in the MS-MPPE-RECV-KEY and MS-MPPE-SEND-KEY attribute fields in the RADIUS message. Because there was no attribute designed to transport keys encrypted, Microsoft created this one that is nowadays used also for other protocols, even if they are not using MPPE at all.

The MS-MPPE fields contains 18 changing bytes, while the rest defines the field as MS-MPPE. RADIUS allows for vendor specific fields and this is one of them, so it has to be clearly defined, because it is not part of the original RFC.

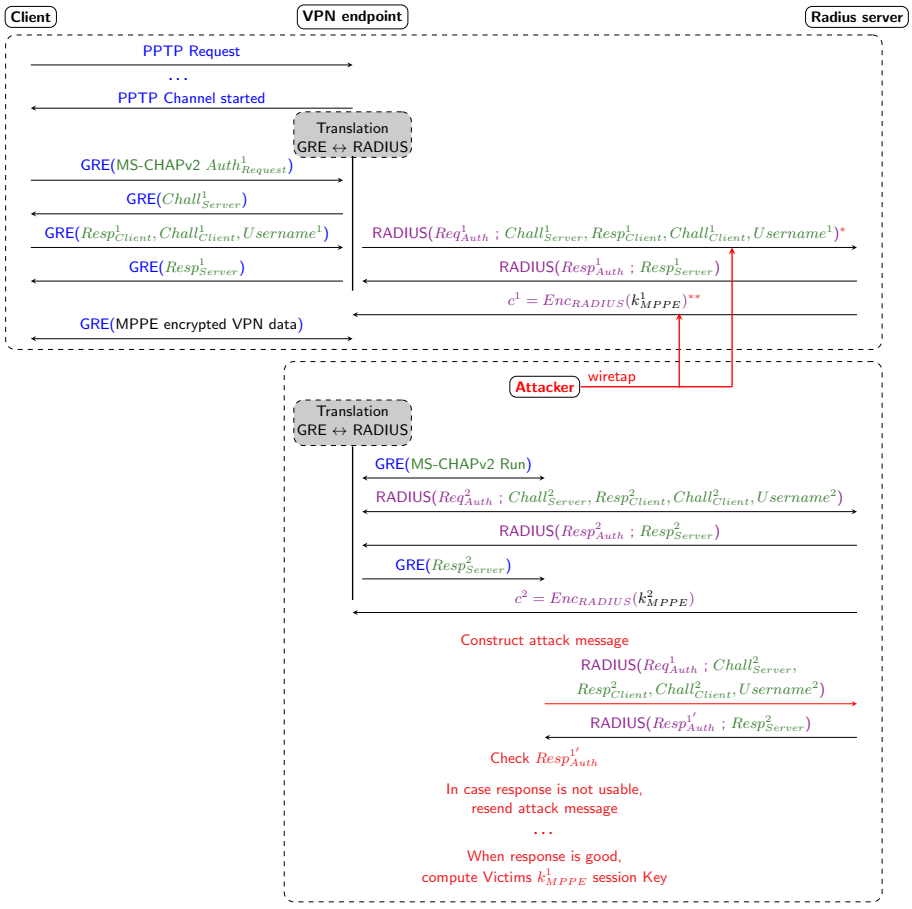

Fig. 4. Attack protocol flow

The 18 bytes contain: 2 bytes Salt and 16 bytes encrypted key. If not the strong version of MPPE, which uses 128 bits keys is used, then the key is even smaller. This means 16 bytes are encrypted using the RADIUS tunnel Encryption Enc_{RADIUS}, while the two bytes Salt are sent in clear. The values can be seen in Figure 4 where the ciphertext C^1 is the encrypted first 16 bytes. The RADIUS encryption would allow to send more blocks for longer keys, as it is used for example in EAP-TTLS but here only one block is used. The attack is shown in Figure 4 and is divided into the following 7 steps:

Step 1: The attacker needs to get himself in the position that he has MitM capabilities on the designated connection between VPN endpoint and RADIUS server.

Step 2: After the attacker is set, he has to wait for the victim user to start a PPTP session and login with MS-CHAPv2 to the VPN through a wiretapped VPN endpoint. In case the potential victim uses another VPN endpoint the attack would not work.

Step 3: The attacker monitors the communication between the VPN endpoint and the RADIUS server.

A device running Wireshark can easily collect all the communication and split them into the different messages that were send. The RADIUS structure dictates that all communication starts with a request from the RADIUS client in this case the VPN endpoint and is answered by a response from the RADIUS server. All following messages are build with the same structure. There is never a response without a request. The attacker is only interested in the last pair of requests and response. If multiple clients connect at the same time to the VPN endpoint, the attacker has to find the correct pair of messages for the desired victim. This information can be read directly from the RADIUS messages, because RADIUS has no privacy protection in place for this information (e.g., $Username$). This step is marked with the wiretap arrows in Figure 4

Step 4: After getting the correct pair of request / response RADIUS messages the attacker takes the Req_{Auth} and Salt from the request and the ciphertext from the $MPPE_{Key}$ field and stores both.

Step 5: The attacker now starts his own login at the same VPN endpoint using his own authentication data. If the messages from the VPN endpoint to the RADIUS server reach again the last pair, the attacker switches the Req^2_{Auth} send from the VPN endpoint to the RADIUS server with the one (Req^1_{Auth}) he stored in the step before. This means that the response will use the old Req_{Auth} instead of the new one. Because the VPN endpoint used another Req_{Auth} it will not be able to decrypt the MS-MPPE key stream k_{MPPE} in the following response, but that is not a problem, because it is not the goal of the attacker to log himself into the network.

Step 6: The attacker compares the Salt that was received in the last response with the Salt stored in step 4. If the Salt is the same he stores the ciphertext again and this time also the plaintext. The attacker has access to the plaintext, because he acts also as a client in the MS-CHAPv2 run and can compute the MPPE-Key k_{MPPE} from internal key material. If the Salt is not the same, the attacker goes back to step 5 and starts there again. The Salt is only 2 bytes (16 bit) long which would offer 65k possible Salts, but the Salt is restricted in different ways (cf. section 2). The first bit is always set to 1, because the message is sent from the server. This protects the encryption from using the client as decryption oracle, but limits the possible Salts. Bits 2 to 5 are used to store the offset. Every time the Salt is used for encryption, the offset is incremented by 1. Due to the fact that two encryptions are done per response from the server (*MS-MPPE-RECV-KEY* and *MS-MPPE-SEND-KEY*), the offset repeats after 8 request and not after 16. As a result the attacker only needs to check every 8th response from the server for the correct Salt. Only the remaining 11 bits are chosen at random, so that after 2048 tries with the same offset the attacker should have found the same random Salt values. On Average an attacker would need 8 * 1024 = 8192 tries.

The Salt was included into the RADIUS encryption to prevent reuse of the RA, but it will show it does not help enough, because the 2048 tries could be done in a short time.

Step 7: The attacker now takes the plaintext and ciphertext he has collected in the last step and computes the XOR of the first 16 bytes of the plaintext called p1 and of

the ciphertext called c1 (cf. Figure 3). We will call this intermediate value X1. This value X1 is now the same as in the stored communication of the victim and in the stored communication of the attacker. This results from the identical Req_{Auth} and Salt. The RADIUS Secret S always stays the same, so that now all input for the MD5 Algorithm is identical, which leads to identical output. The attacker now can XOR the value X1 with the first block of the victims C^1. The result are the first 16 plaintext bytes, which is the MPPE-Key k_{MPPE} of the victim.

3.4 Practical Evaluation

Setup. The setup consists of four different Virtualbox machines, one acting as the client, one as the VPN endpoint, one as the RADIUS server and the last machine performing the attack. As the operating system *Ubuntu Linux 15.04* with Kernel *3.19.0-39-generic* was used. The client had *pptp-linux 1.7-2.7* installed for the pptp client. The VPN endpoint used *pppd 2.4.6* as PPTP server and the *FreeRadius Client 1.1.7* for the connection to the RADIUS server. The attack was tested against the most used RADIUS server FreeRADIUS. We used the server with version 3.0.10 in the default configuration, without any further modifications. The host machine was equipped with an Intel Core i5-6600K @ 3.5 GHz and 16 Gbyte of DDR4 memory.

Results. The attack was run for 432 times on the attack machine (see. section 3.4). The evaluation showed that for these 432 attack runs the average time was 62 seconds and it took on average 18847 protocol runs until the Salt was correct. The theoretical average is 2^{14} = 16384. This shows that the random salts generated by the RADIUS server are chosen as random as roughly expected. For this test we ran 8,1 Million protocol runs in total. Our cross protocol attack thus reduced the complexity by 2^{42} compared to the attack of Schneier et. al [13] and Marlinspike [7]. Still today brute-forcing DES keys on an FPGA cluster takes an average of 7 hours (25,200 seconds) and hardware costs of around $ 140,000 [3]. This means, our attack speeds up the time for the decryption process by factor of 124, compared to the traditional brute-force approach and is achievable on standard computer hardware, which cost around $ 700 US (reduction by factor 200).

4 Conclusion

In the paper we showed two novel attacks. A known-plaintext attack on RADIUS encryption and a chosen-ciphertext attack on PPTP VPN. We describe how both of these attacks can be combined in a cross protocol attack to decrypt PPTP VPN sessions. Analyzes of public VPN services providers showed that PPTP is used by 60% of them [14]. Further, all Windows and Linux operating systems currently support PPTP VPNs. For our attacks we give a description of PPTP VPN and how they are used together with MS-CHAPv2 and RADIUS server. Based on the description we show how an attacker can use our cross protocol attack to decrypt PPTP VPN sessions in a realistic scenario in one minute. In contrast to previous published attacks [7,13], we reduced the computational complexity by factor 2^{42}. The decryption of a PPTP VPN session is now

achievable with just 2^{14} protocol runs. Our attacks do not need special hardware and run on every modern device, from standard computers down to smartphones. Further, the attack needs on average 62 seconds, leading to a time reduction by the factor of 124, compared to the brute-force of the complete DES key space, which also required special cracking hardware [3].

We saw with this attack that even after 18 years a protocol is in the wild improvements for attacks can be found, by taking a look on cross protocol usage. At first we tried to drive the attack on EAP-(T)TLS implementations [2], but realized during the testing, that an additional MAC over the whole message is computed preventing our attack.

In the RFC 2868 [17], which specified the used RADIUS attributes, multiple protocols are defined, we analyzed only PPTP. Similar attacks may be possible on other protocols like L2TP, depending on the used key derivation function and are worth looking at in future research.

References

1. The Point-to-Point protocol (PPP). RFC 1661, IETF (Jul 1994)
2. Aboba, B.D., Calhoun, P.: RADIUS (remote authentication dial in user service) support for extensible authentication protocol (EAP). RFC 3579, IETF (Sep 2003)
3. Amy, V.: The state of the art in key cracking (2016), `https://www.voltage.com/breach/the-state-of-the-art-in-key-cracking/`
4. Eisinger, J.: Exploiting known security holes in microsoft's pptp authentication extensions (ms-chapv2). University of Freiburg,[cit. 2008-27-05] Dostupné (2001)
5. Hamzeh, K., Pall, G., Verthein, W., Taarud, J., Little, W., Zorn, G.: Point-to-Point tunneling protocol. RFC 2637, IETF (Jul 1999)
6. Hanks, S., Li, T., Farinacci, D., Traina, P.: Generic routing encapsulation (GRE). RFC 1701, IETF (Oct 1994)
7. Marlinspike, M.M., Hulton, D., Ray, M.: Defeating pptp vpns and wpa2 enterprise with ms-chapv2. Defcon, July (2012)
8. Ornaghi, A., Valleri, M.: Man in the middle attacks demos. Blackhat 19 (2003)
9. Pall, G., Zorn, G.: Microsoft Point-To-Point encryption (MPPE) protocol. RFC 3078, IETF (Mar 2001)
10. Paterson, K.G., Poettering, B., Schuldt, J.C.: Big bias hunting in amazonia: Large-scale computation and exploitation of rc4 biases. In: Advances in Cryptology–ASIACRYPT 2014, pp. 398–419. Springer (2014)
11. Project, F.S.: Freeradius server, `http://freeradius.org`
12. Rigney, C., Willens, S., Rubens, A., Simpson, W.: Remote authentication dial in user service (RADIUS). RFC 2865, IETF (Jun 2000)
13. Schneier, B., Mudge: Cryptanalysis of microsoft's point-to-point tunneling protocol (pptp). pp. 132–141. CCS (1998)
14. Site, T.O.P.: Detailed vpn comparison chart, `https://thatoneprivacysite.net/vpn-comparison-chart/`
15. Zorn, G.: Microsoft PPP CHAP extensions, version 2. RFC 2759, IETF (Jan 2000)
16. Zorn, G.: Deriving keys for use with microsoft Point-to-Point encryption (MPPE). RFC 3079, IETF (Mar 2001)
17. Zorn, G., Leifer, D., Rubens, A., Shriver, J., Holdrege, M., Goyret, I.: RADIUS attributes for tunnel protocol support. RFC 2868, IETF (Jun 2000)