COM-506 Seminar handout: Dragonblood

Based on: Dragonblood: Analyzing the Dragonfly Handshake of WPA3 and EAP-pwd (Mathy Vanhoef and Eyal Ronen)

Nicolas Dutly[1]

EPFL, Switzerland, nicolas.dutly@epfl.ch

Abstract. This handout provides a short summary of the COM-506 seminar presentation on the Dragonblood paper by Mathy Vanhoef and Eyal Ronen describing various attacks against WPA3's dragonfly (also known as SAE) protocol.

Keywords: COM-503 · Dragonblood · WPA3

Introduction

The security protocols described in IEEE 802.11 for securing wireless communications have long been the subject of extensive scrutiny. The forerunner of wireless security, WEP (wired equivalent privacy), was introduced in 1997. Serious flaws in the way RC4 was used lead to attacks which could easily recover the key [1]. WPA followed as an interim protocol fixing the RC4 flaws found in WEP. WPA2 was introduced in 2004, mandating support for AES-CCMP. To date various flaws have been found in WPA2 such as the KRACK attack in 20017 [2]. Furthermore WPA2-PSK is by design vulnerable to offline bruteforce attacks. Its successor, WPA3 aims to solve this shortcoming by implementing the SAE (simultaneous authentication of equals) protocol which is a password authenticated key exchange (PAKE). It is a variant of the *dragonfly* protocol from which the Dragonblood paper takes its name. Using this protocol allows the conversion of a low entropy shared secret (such as a common password) to a high entropy shared key. Unfortunately several vulnerabilities have been discovered in the SAE protocol which we will summarize here. For an in depth analysis we refer to the original Dragonblood paper [3].

Password derivation and side channel attacks

The MODP case

The first step in the SAE protocol is to convert a low entropy password p^* to an element e of a multiplicative group modulo p (MODP group) or a point on an elliptic curve $E \in E(\mathbb{F}_p)$ in the case of elliptic curve cryptography. In the case of MODP groups the method is called *hash-to-element*. A pseudocode implementation of the method is presented bellow:

```
def hash_to_group(password, mac1, mac2):
        for counter in range(1,256):
            seed = Hash(mac1,mac2,password,counter)
            value = KDF(seed, label, p)
            if value >= p: continue
            Elem = math.pow(value, (p-1)/q) mod p
            if Elem > 1: return Elem
```

The algorithm instantiates a seed which depends on the mac addresses of both parties and the low entropy password. A key derivation function is then used to produce a value which is then converted to an element of the MODP group. It is possible that the KDF produces a value larger or equal to p, which cannot be converted to a valid MODP element. In that case the algorithm tries again in the next iteration (the seed depends on the current iteration). Computing KDF$(\cdot)$

mod p would not be a suitable replacement to this try-and-increment strategy due to the induced bias it would cause. This detail is important when we will discuss the case of hashing to an element of an elliptic curve.

We can see that in the above code the number of iterations directly depends on the password and the two mac addresses. This timing leak can be exploited to deduce how many iterations was needed for a specific mac and password pair. By simulating the number of iterations needed for a given password and mac pair[1] we can iteratively prune passwords from a given list. Namely if a password p_1 from the list causes a difference in simulated and measured iterations it implies that p_1 cannot have been the real password.

The EC case

In the case of elliptic curves the implemented *hash-to-curve* method is slightly different and actually includes a number of defenses against timing leaks:

```
def hash_to_curve(password, mac1, mac2):
    k = 40, found = False
    while count < k:
        count++
        seed = Hash(mac1,mac2,password,count)
        value = KDF(seed, label, p)
        if value >= p: continue
        if quad_res(value^3 + a * value + b, p):
            if not found:
                x,found = value, True
                password = rand()
    y = sqrt(x^3+a*x+b) mod p
    return (x,y)
```

We can see that the number of iteration is fixed and does not depend on attacker controlled data. Unfortunately, there are some subtle points to take into account which can still leak information if the constant k is not high enough[2]. First, the function `quad_res` checks whether $v^3 + av + b$ is a quadratic residue modulo p. If yes this implies that there is some c such that $v^3 + av + b \equiv c^2 \mod p$. This implies the existence of an element y such that (v, y) is a valid point on the elliptic curve. This procedure can leak information [4]. To prevent this a method called QR blinding is employed which multiplies the number to be checked by a random square r^2 and then multiplies the result by a non-residue n. This is important to keep in mind as it increases the computational cost of the password derivation process (which cannot be done offline since it includes the MAC addresses of the peers). This will be a decisive factor when discussing denial of service attacks.

Exploiting the timing leak in the hash-to-curve method is non trivial as the number of iterations depends on (excluding mac and counter values):

1. The real password
2. The randomly generated password in the extra iterations

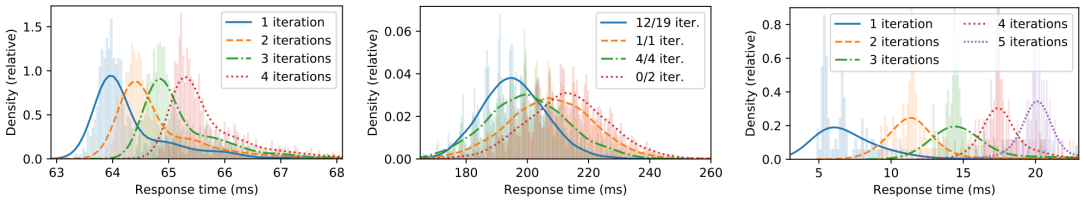
Nonetheless, the execution time variance is still dependent on when the password is found. Interestingly it was found that brainpool curves are more vulnerable to timing leaks than NIST curves. This is due to the structure of the primes used by both groups of curves. In the NIST case the primes are *quasi-Mersenne*, which means that they can be written as a sum of powers of two minus one. For example:

$$p_{192} = 2^{192} - 2^{64} - 1$$

$$p_{521} = 2^{521} - 1$$

The KDF always output n random bits with n being the number of bits needed to represent a prime. Due to the special structure of NIST primes, the probability that the KDF returns an

[1] the client's MAC address can easily be spoofed
[2] The Wifi alliance currently recommends $k = 40$



(a) WPA3 AP with MODP group 22. (b) WPA3 AP with Brainpool curve 29. (c) EAP-pwd client with curve P-256.

Figure 1: Being able to determine the number of iterations in the hash-to-group and hash-to-curve method allows an attacker to fingerprint and exclude possible passwords [3] (The EAP-pwd protocol does not implement fixed iteration which explains the leaks despite the usage of a NIST curve)

invalid output is minimal (for example it is 2^{-521} for p_{521}). This is not the case for brainpool curves. For example, with the curve brainpoolP384r we have a 45.03% probability of getting an invalid value.

In both the MODP and EC case it is thus possible to determine the number of iterations which was likely executed. As mentioned earlier, this allows us to fingerprint and exclude / prune passwords. Besides the timing side channels presented here it is also possible to achieve similar results using cache-based side channels. As this requires a stronger attacker model and in the interest of brevity we chose to not go into further details. We refer the interested reader to [3].

Denial of Service attacks

The keen reader will have noticed that the hash-to-curve method presented above is computationally expensive. This is mostly due to the side channel defenses such as looping for k fixed iterations and the implementation of QR blinding which requires additional multiplication and random number generations. Initially a secret cookie system was intended to prevent potential denial of service attacks by flooding the AP with commit frames (similar to wireguard's defense [5]). But due to the broadcast nature of wireless networks these cookies can easily be captured and reflected, bypassing said defenses. Spoofing eight commit frames per seconds causes a CPU usage of up to 80% on professional grade access points.

WPA3 transition mode downgrades

The WPA3 transition mode was introduced in order to smooth the transition to a WPA3 only world. It mandates that networks running in WPA3 transition mode support both WPA2 and WPA3 using the same passphrase. By default WPA2 support the detection of downgrade attempts by including an authenticated RSNE elements which specify the ciphers and authentication mechanisms which are supported in the four way handshake. To circumvent this an attacker can create a rogue AP with the same SSID and broadcast WPA2-only support. The first message of the four way handshake is non-authenticated, i.e forgable by the attacker. The client will then respond with an authenticated message which can then be used to bruteforce the password in a standard manner. Note that this attack forces WPA3 capable clients to use WPA2. A solution to this would be to remember which networks support WPA3, similar to the way HSTS works.

Reflection and invalid curve attacks

We will now shortly mention a couple of implementation specific vulnerabilities of the SAE protocol. The first vulnerability mainly affects the EAP-PWD protocol (used in enterprise networks) which also uses the dragonfly protocol. In EAP-PWD the AP initiates the handshake, sending the first commit message. An attacker can reflect that message and later reflect the AP's confirm message leading to a successful authentication in many EAP-PWD implementations. Note that this does not allow the attacker to encrypt or decrypt traffic so the impact is somewhat limited.

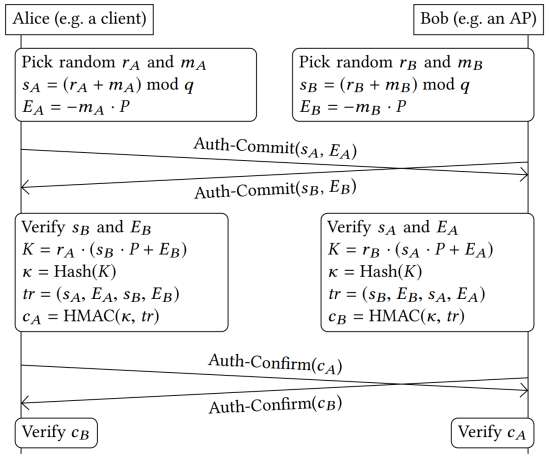


Figure 2: The SAE handshake consists of two phases (after a common password has been derived): The commit and confirm phases. The figure illustrates the case where elliptic curves are used. As such the derived password P is a point on a curve $e \in E(\mathbb{F}_p)$

The second vulnerability also targets the SAE handshake: An attacker can send a point P in the first commit message which is non-valid. Buggy implementations do not verify the validity of the received point. This causes the AP to derive a predictable key which the attacker can guess.

Conclusion

Even with the problems presented in the summary, the usage of WPA3 over WPA2 is still recommendable. Most of the presented problems have been fixed. Some problems, such as the high overhead of the side channel mitigations in the hash-to-curve method are inherent to the dragonfly protocol. It is important to note that other hash-to-curve methods do exist (such as SWU, Icart) which require significantly less operations than the method used in the dragonfly protocol.

References

[1] Scott R. Fluhrer, Itsik Mantin, and Adi Shamir. Weaknesses in the key scheduling algorithm of rc4. In *Revised Papers from the 8th Annual International Workshop on Selected Areas in Cryptography*, SAC '01, page 1–24, Berlin, Heidelberg, 2001. Springer-Verlag.

[2] Mathy Vanhoef and Frank Piessens. Key reinstallation attacks: Forcing nonce reuse in WPA2. In *Proceedings of the 24th ACM Conference on Computer and Communications Security (CCS)*. ACM, 2017.

[3] Mathy Vanhoef and Eyal Ronen. Dragonblood: Analyzing the Dragonfly handshake of WPA3 and EAP-pwd. In *IEEE Symposium on Security & Privacy (SP)*. IEEE, 2020.

[4] Thomas Icart. How to hash into elliptic curves. Cryptology ePrint Archive, Report 2009/226, 2009. https://eprint.iacr.org/2009/226.

[5] Jason A. Donenfeld. Wireguard: Next generation kernel network tunnel. *Proceedings 2017 Network and Distributed System Security Symposium*, 2017.