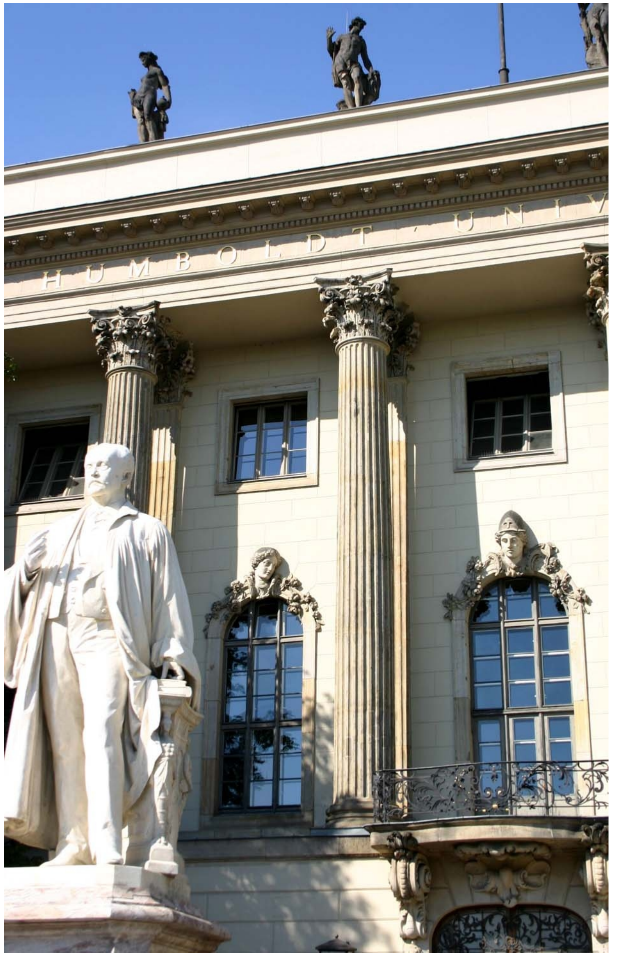
IT Security Workshop 2018

WPA3 Dragonfly Handshake – SAE Handshake

Nikolai Tschacher



Freitag 12.10. 2018
Prof. J.P.Redlich, Dr. W.Müller

Why a new handshake?

- WPA2 handshake *is still secure,* but susceptible to offline dictionary attacks and has no forward secrecy (once a key is cracked, all old captured traffic may be decrypted)

- Old 4-Way-Handshake from WPA2 is still used, dragonfly yields a strong Pairwise Master Key (PMK) to be piped into 4-Way-HS

- Dragonfly prevents offline dictionary attacks and ensures perfect forward secrecy

What was my task?

- To see what happens when trying to implement the new WPA3 Dragonfly / SAE Handshake (rfc7664).

- Learn from the implementation process and find ways to pentest/fuzz implementations that will emerge soon

- Python3 and ECC with the brainpoolP256r1 curve.

Implementation

- https://github.com/NikolaiT/Dragonfly-SAE

- Works! I took the curve brainpoolP256t1 from rfc5639

- Advantage: Implementing the protocol shows mistakes that are easy to make using rfc7664. Good for follow up studies...

- Reasonable to expect that vendors are going to deviate from the RFC as well

Dragonfly Key Exchange

Two parties start with a shared (salted) password that was established in an out-of-band mechanism

Agree to a domain parameter set for ECC (also FFC possibly)

Deterministic method to derive a secure key (PE) with a "hunting-and-pecking" technique

Any method of deterministically mapping a secret string into an element in a selected group may be used (should include MAC address) → too many DOF?

Deriving the PE

Compute a hash based on the mac addresses of the participants + the shared password + counter.

This hash is used as a seed for a KDF. The KDF is basically a RBG that stretches the input to a length of len(p) + 64 and yields a random number.

This number is used as the x-coordinate for the curve equation. Then we check whether this is a quadratic residue (Legendre Symbol). If its not, increase the counter by 1 and try again

Quick reminder: $y^2 \equiv x^3 + a*x + b \pmod{p}$

The Commit Exchange

Each peer chooses two random numbers, *private* and *mask* in the domain of p

scalar = (private + mask) modulo q

element = inverse(scalar_mul(mask, PE))

q is a prime and the order of the generator G and thus the size of the cryptographic subgroup

The peers exchange their scalar and elements and the shared secret is derived:

```
ss = scalar_mul(private,
                point_add(peer_element,
                          scalar_mul(peer_scalar, PE)))
```

The Confirm Exchange

The shared key is stretched in two parts both having the length of the order of the group: K = kck | mk

Each peer creates a confirm token and sends this hash to the other peer. Both parties can confirm that they possess the same kck and thus that they have the same shared secret ss.

confirm = H(kck | scalar | peer_scalar |
element | peer_Element | MAC)

Pairwise Master Key = H(kck | scalar + peer_scalar mod q)

Implications

PMK is based on the random scalar that was chosen in the commit exchange and not on the pre shared password! This implies <u>perfect forward secrecy</u>

The password only determines the starting point in the group!
Public_Key = random * Password

The commit exchange is a sort of ECDH that is additionally authenticated.

This means that a successful handshake implies that each peer has the same pre shared password.

Offline Attack?

In WPA2, an attacker can deauthenticate a STA and sniff the handshake and launch an offline dictionary attack

With Dragonfly this is not feasible, since sniffing the *scalar* and *element* does not enable us to compute *private* or *mask*

Without those values, an attacker cannot verify if a guessed password is correct

The attacker <u>must</u> participate in a handshake with a peer to make a guess at the password (Rate limiting easily implementable)

Side Channel Attacks

The hunting and pecking technique must have a fixed amount of iterations to thwart side channel attacks

When using big curves, implementation may become slower and susceptible to DOS attacks

Fuzzing Possibilities

Dragonfly/SAE can be initiated by either side

Either peer can commit at any time, a peer can confirm only after it has committed and the other peer has committed, a peer can accept after its peer has confirmed and the confirmation has been verified……

Many different states to keep track on the peers!

A fuzzer might create different state combinations and see how implementations react

random.seed()

While implementing the function to derive the password, I seeded the PRBG with the hashed password as suggested in the RFC: random.seed(password_hash)

“Function KDF is a key derivation function (see, for instance, [SP800-108] Chen, L., "Recommendation for Key Derivation Using Pseudorandom Functions")“

BUT later we need to create two random numbers in the commit exchange!

```
self.private = random.randrange(1, self.p)
self.mask = random.randrange(1, self.p)
```

When the password_hash is leaked, the whole dragonfly exchange is broken because one can predict the generated random numbers!

What I do, is obviously flawed: random.seed(time.process_time()) or random.seed()

Conclusion

Implementing a new protocol is hard

Rfc7664 leaves many degrees of freedom for implementations

Hash function free to chose

Hunting and pecking very flexible

Key Derivation Function is not accurately specified: “The base is then stretched using the technique from Section B.5.1 of [FIPS186-4].”

→ But FIPS186-4 leaves a lot of flexibility

Refs

Introduction to WPA3:
https://wlan1nde.wordpress.com/2018/09/14/wpa3-improving-your-wlan-security/

RFC behind the dragonfly handshake that is used in WPA3:
https://tools.ietf.org/html/rfc7664

Update of the most recent changes in the WPA3 certification program:
https://www.mathyvanhoef.com/2018/06/wpa3-missed-opportunity.html

Bruce on WPA3: https://www.schneier.com/blog/archives/2018/07/wpa3.html

Discussion on problems with dragonfly:
https://news.ycombinator.com/item?id=17403697

Author of Dragonfly on WPA3 himself:
https://blogs.arubanetworks.com/industries/wpa3-the-next-generation-in-secure-mobility/

WPA2 4-Way Handshake: https://wlan1nde.wordpress.com/2014/10/27/4-way-handshake/