WPA2 Enterprise and Wi-Fi security

Pierre Bourdon

delroth@lse.epita.fr
http://lse.epita.fr

July 20, 2012

- Also known as IEEE 802.11
- First commercial use in 1999
- Omnipresent for LANs
- Currently (kind of) secure

Well known security mechanisms

- WEP: broken in 2001 (24 bits IV for RC4)
- WPA-PSK: broken in 2010 (QoS based attack, only on TKIP)
- WPA2-PSK: not yet broken

PSK vs Enterprise

- PSK is also known as WPA Personal
- WPA Enterprise provides authentication and security against access point spoofing
- Secure if you configure it properly
- We're going to look at an attack on WPA2 Enterprise based on a configuration issue

Goal

- Provide a way for enterprise users to know *who* is connected to the access point
- Can be used to link Wi-Fi users to system users (same password, SSO, ...)
- Passwords can be replaced with better mechanisms (smartcards for example)

RADIUS and EAP

- RADIUS is a standard authentication protocol that can be used for centralized user account handling
- EAP is a protocol framework used to communicate with a RADIUS server
- Several protocols based on EAP with vendor extensions
- In WPA Enterprise the access point tunnels EAP requests to the RADIUS server (which isn't accessible from another host)

Most common EAP types

- LEAP, created by Cisco, broken in 2004 (ASLEAP)
- EAP-MD5, standard, prone to MITM, weak hash function
- EAP-PSK, which requires an additional key
- EAP-IKEv2, used for asymmetric credentials (smartcards)
- EAP-TLS, which authenticates with a TLS client certificate
- PEAP, a way to encapsulate another EAP type to avoid MITM

PEAP

- Created by Cisco, Microsoft and RSA, available since WinXP
- EAP tunnelled in TLS
- The client can check that the RADIUS server provides a valid certificate and it can avoid MITM attacks that way
- As secure as your PKI is

EAP-MSCHAPv2

- Handles authentication using MSCHAPv2, a Microsoft protocol
- Basically proven unsafe since 1999 (*Cryptanalysis of Microsoft's PPTP Authentication Extensions*, Bruce Schneier)
- Uses the NTLM hashing algorithm, which is based on MD4
- Still the most used EAP protocol because nothing better exists at the moment

Access point spoofing

- Wi-Fi has no protection against access point spoofing
- Just announce the same SSID
- If you emit a better signal clients will prefer your AP
- Should cause no problem because authentication should fail after association

EAP-TLS and PKI

- TLS is only MITM-proof if you use certificates signed by a common CA
- Lots of built-in CA in operating systems that you must pay in order to get a certificate
- If you control your computer deployments you can add a builtin CA to the devices
- An invalid certificate should be a fatal error and clients should never send their authentification requests through a non validated EAP tunnel

What shouldn't be done - Windows

Dans la fenêtre qui vient de s'ouvrir (**Image 07**), premièrement (1), décocher "Valider le certificat du serveur".
Deuxièmement (2), vérifier que la méthode d'authentification sélectionnée est bien "Mot de passe sécurisé (EAP-M
Troisièmement (3), cliquer sur "Configurer...".

What shouldn't be done - OSX

Dans "Nom d'utilisateur", entrer votre login.

Dans "Mot de passe", entrer votre mot de passe socks.

Pour finir, une fenêtre "Vérifier le certificat" apparait. Cliquez simplement sur "Continuer"

What shouldn't be done - Fedora

Pour finir, cliquer sur « Connect ». Une nouvelle fenêtre annonçant « No Certificate Authority certificate chosen » apparait. Il n'y a effectivement plus besoin de certificats pour établir une connexion wifi : cliquer sur « Ignore ».

MSCHAPv2 "security"

- Full of holes, proven unsafe in 1999, proven unsafe again in 2001
- Bruce Scheneier showed in 1999 that most of the protocol was completely useless, which is a sign that the inventors were clueless
- Jochen Eisinger discovered in 2001 that MSCHAPv2 leaked the 2 last bytes of the NTLM hash in the handshake, even with all the cryptography trying to avoid it
- Getting the last two bytes of the hash requires you to compute 2^{16} DES hashes, which can be done in less than a second on any modern computer
- You can reduce your password search size by 2^{16} by computing the NTLM hash first then only checking it if the last two bytes match

Password size

- Hypothetical case: 8 characters, charset size is 88
- $88^8 = 3596345248055296$ possible passwords
- If you can test 3 billion passwords per second, 14 days max, 7 days average
- If you can test 10 billion passwords per second, 4 days max, 2 days average
- Spoiler: 3 billion passwords is what my $100 GPU can compute
- Use passphrases and/or password managers and/or private keys

What do we need?

- Access point spoofing
- RADIUS handshake sniffing
- Password cracking

Don't try this at home

- If you fail to set up your spoofing properly you will piss off everyone around you who will not have access to the internet
- AeroHive Wi-Fi emitters have a built-in rogue access point triangulation system
- Your BSSID is visible by anyone and attempts to connect to the network with this BSSID later on can be detected
- It's not very useful anyway

hostapd

- Go-to tool to make a Wi-Fi access point under Linux
- Very configurable: country code, security type, etc.
- Supports WPA2 Enterprise and connecting to a RADIUS server
- Very easy to configure, documented example config

FreeRADIUS

- Best open source RADIUS server implementation
- Supports PEAP, MSCHAPv2 as well as a lot of other EAP types
- Can be used alongside OpenLDAP if you want
- Again, very well documented, not as easy to configure as hostapd

FreeRADIUS-WPE

- Developed by the guys who broke the LEAP protocol and wanted to intercept RADIUS handshakes
- Patch for FreeRADIUS which adds automatic logging of connection attempts in an easy to use format
- Saves a lot of time: FreeRADIUS is a large codebase

```
*** mschap: Fri Jul  6 12:05:03 2012
username: bourdo_p
challenge: f8:e0:a4:86:...
response: c8:ef:50:36:6a:3d:ca:8f:89:41:d3:...
```

Reference implementation

- First implementation of the MSCHAPv2 cracking algorithm in Python
- Slow proof of concept used to test if the algorithm worked or not
- Takes 20min for a 6 characters password...

Reference implementation

```python
def bruteforce(challenge, response, charset, size):
    l2 = find_last_two_bytes(challenge, response)
    if l2 is None:
        raise RuntimeError("wtf")
    for password, hash in all_possible_passwords(charset, size, l2):
        print("Trying %r ..." % password)
        if des_encrypt(challenge, hash[0:7]) == response[0:8]:
            if des_encrypt(challenge, hash[7:14]) == response[8:16]:
                return password
```

OpenCL rewrite

- GPUs are good for password cracking, right?
- Not that much for DES though
- Just make `all_possible_password` run on GPU and the DES checks on CPU!
- About 250 lines of OpenCL, 600 lines of Python

Performance

- On an AMD HD6770 ($100), 2.6B of NTLM are tested each second
- On average that means the CPU tests only 40K passwords with DES, so the CPU is not a bottleneck at all
- With OpenCL running on an Intel C2D CPU @ 2.6GHz, 100M NTLM/s

Adding distribution

- One computer is too slow (14 days max)
- Solution: throw more computers at the problem!
- Rewrote the cracking tool to be able to take orders from a centralized distribution server
- Almost no performance cost (every network access can be done asynchronously)
- The server itself is about 350 lines of Python

Cloud computing

- Where could we find more computers?
- Aren't we in a computer engineering school?
- Large computer labs with 80 core 2 duo computers are equivalent to 4 $100 GPU on this problem
- ... but we have 3 labs like that
- We were able to reach a maximum cracking time of 2 days, average 1 day

Conclusion

- WPA2 Enterprise can be deployed in a secure way
- MSCHAPv2 is broken and there is no good alternative as of now
- 8 character passwords are a thing from the past, go for longer passwords (passphrases!)

Questions?

- @delroth_
- delroth@lse.epita.fr
- http://intra-bocal.epitech.eu/