# Offline bruteforce attack on WiFi Protected Setup

Dominique Bongard
Founder
0xcite, Switzerland
@reversity

# Agenda

- Introduction to WPS
- WPS PIN External Registrar Protocol
- Online Bruteforce attack on WPS PIN
- Offline Bruteforce attack on WPS PIN
- Vendor reponses
- Bonus

# WARNING

This presentation may contain illustrations by Ange Albertini

# What is WPS ?

- Wi-Fi Protected Setup (WPS) or Wi-Fi Simple Configuration (WSC)
- „A specification for easy, secure setup and introduction of devices into WPA2-enabled 802.11 networks"
- Offers several methods for In-Band or Out-of-Band device setup
- Severely broken protocol!
- The technical specification can be purchased online for $99
- Some old versions can be found floating on the net

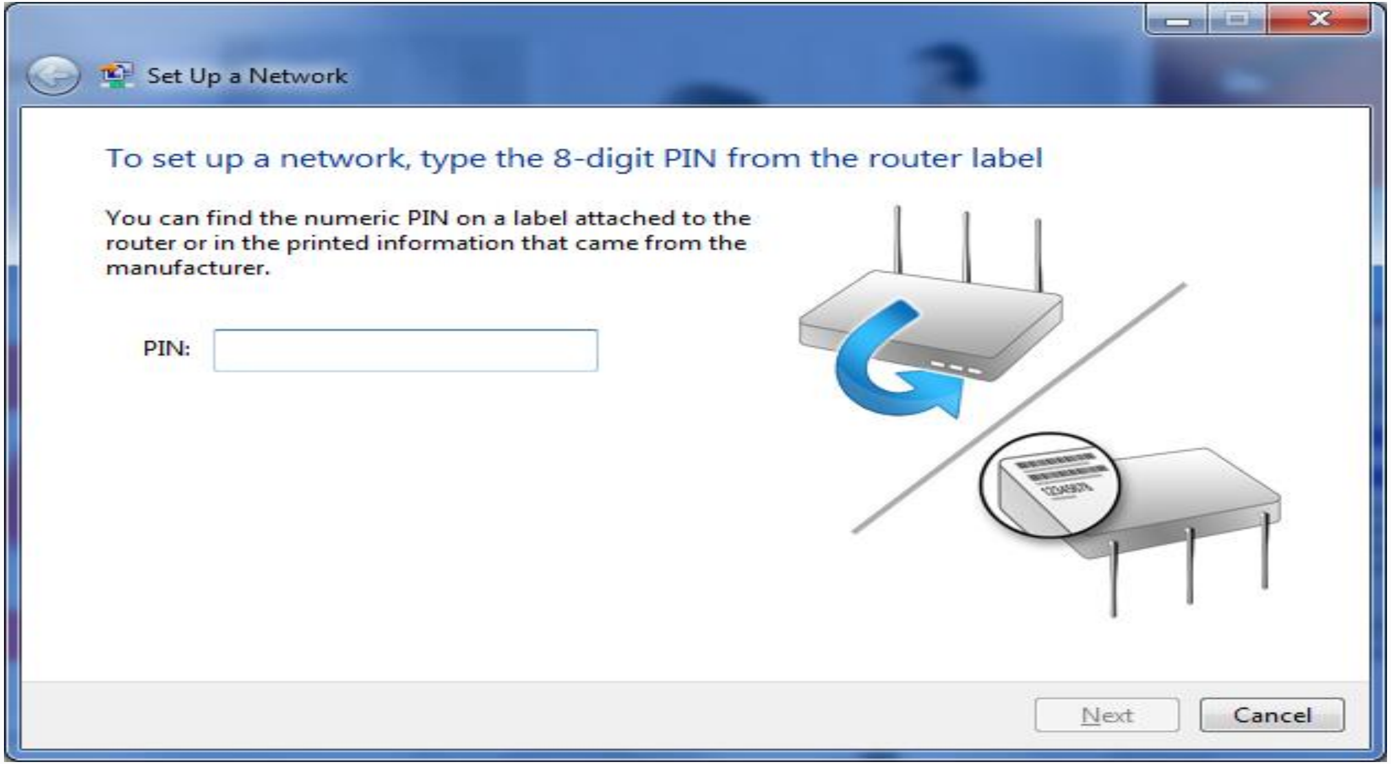
# Example :WPS PIN protocol (Headless)

S/N: XXXXXXXXXXXX

MAC ID: YYYYYYYYYYYY

WPS PIN: ZZZZZZZZ

# Example: WPS PIN protocol



# WPS Configuration Methods

- USB Flash Drive (Deprecated)
- Ethernet (Deprecated)
- **Static PIN on device label**
- Display
- NFC Token
- **Push Button**
- Keypad

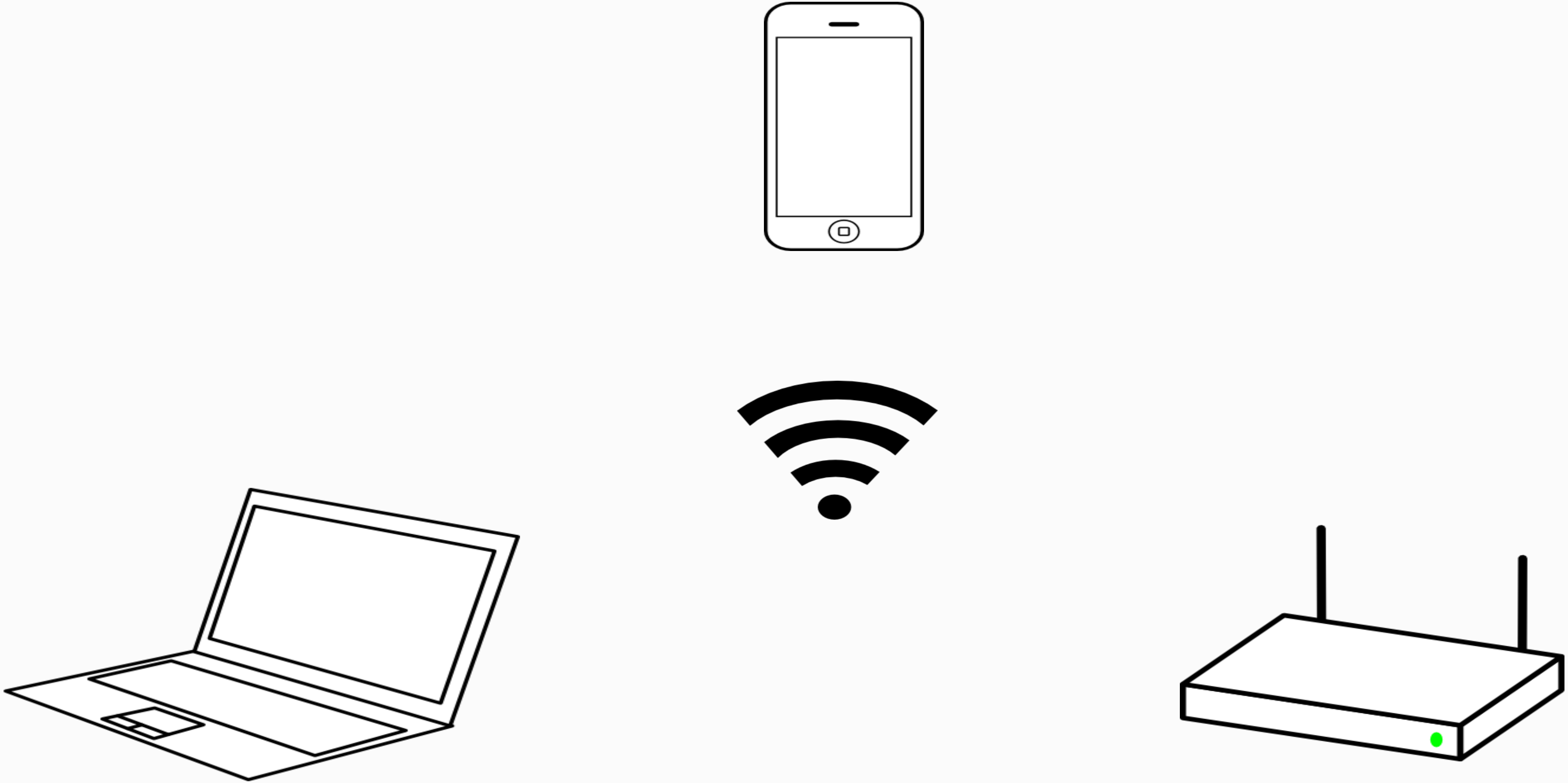
# WPS Misconception 1

- To register with WPS you **don't** need to know the PIN **and** press the WPS button
- You need to know the PIN **OR** press the WPS button

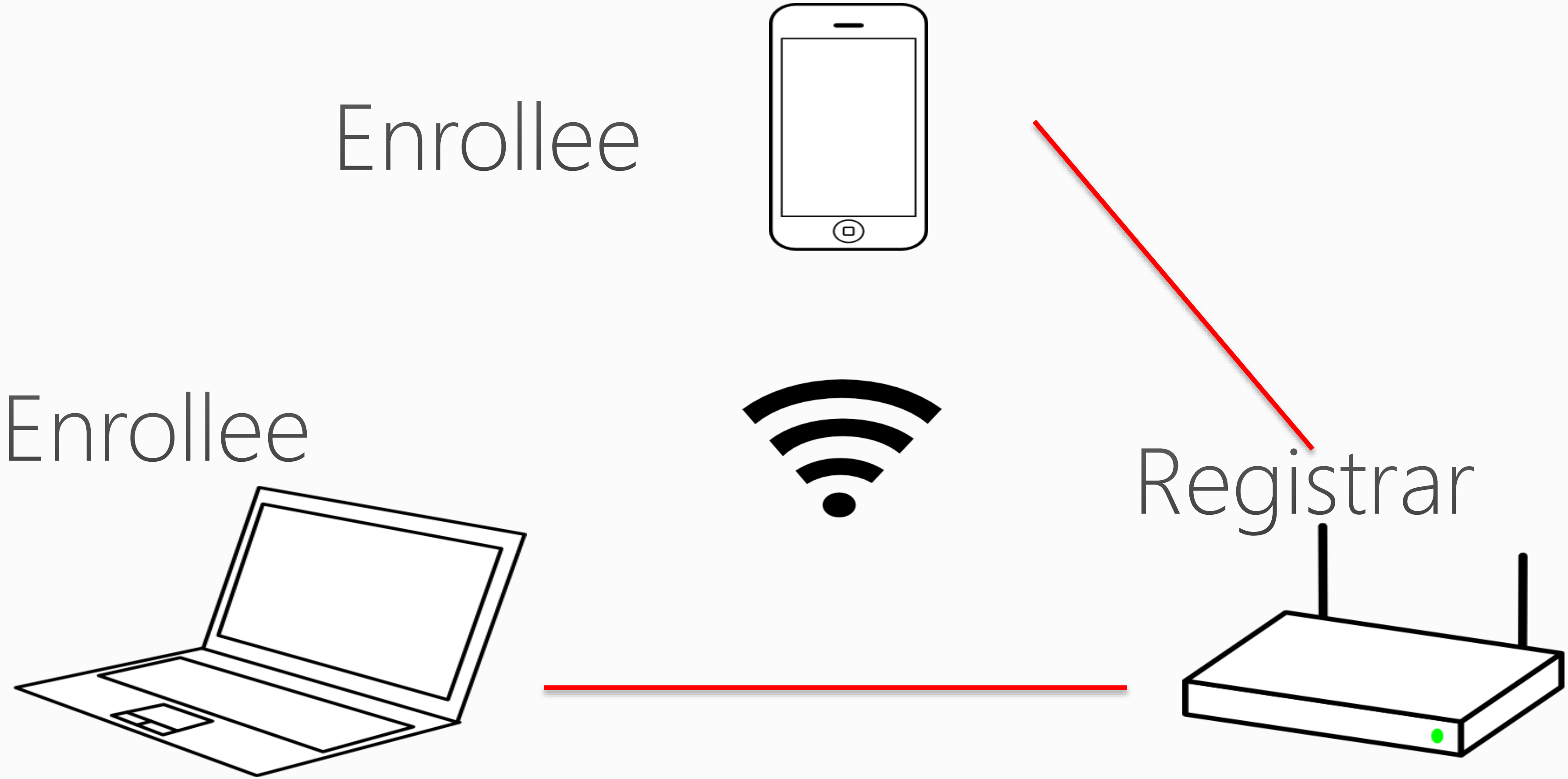
# WPS Definitions

- **Enrollee** : A device seeking to join a WLAN domain
- **Registrar** : An entity with the authority to issue WLAN credentials
- **External Registrar** : A registrar that is separate from the AP
- **AP** : An infrastructure-mode 802.11 Access Point
- **Headless Device** : A device without a screen or display

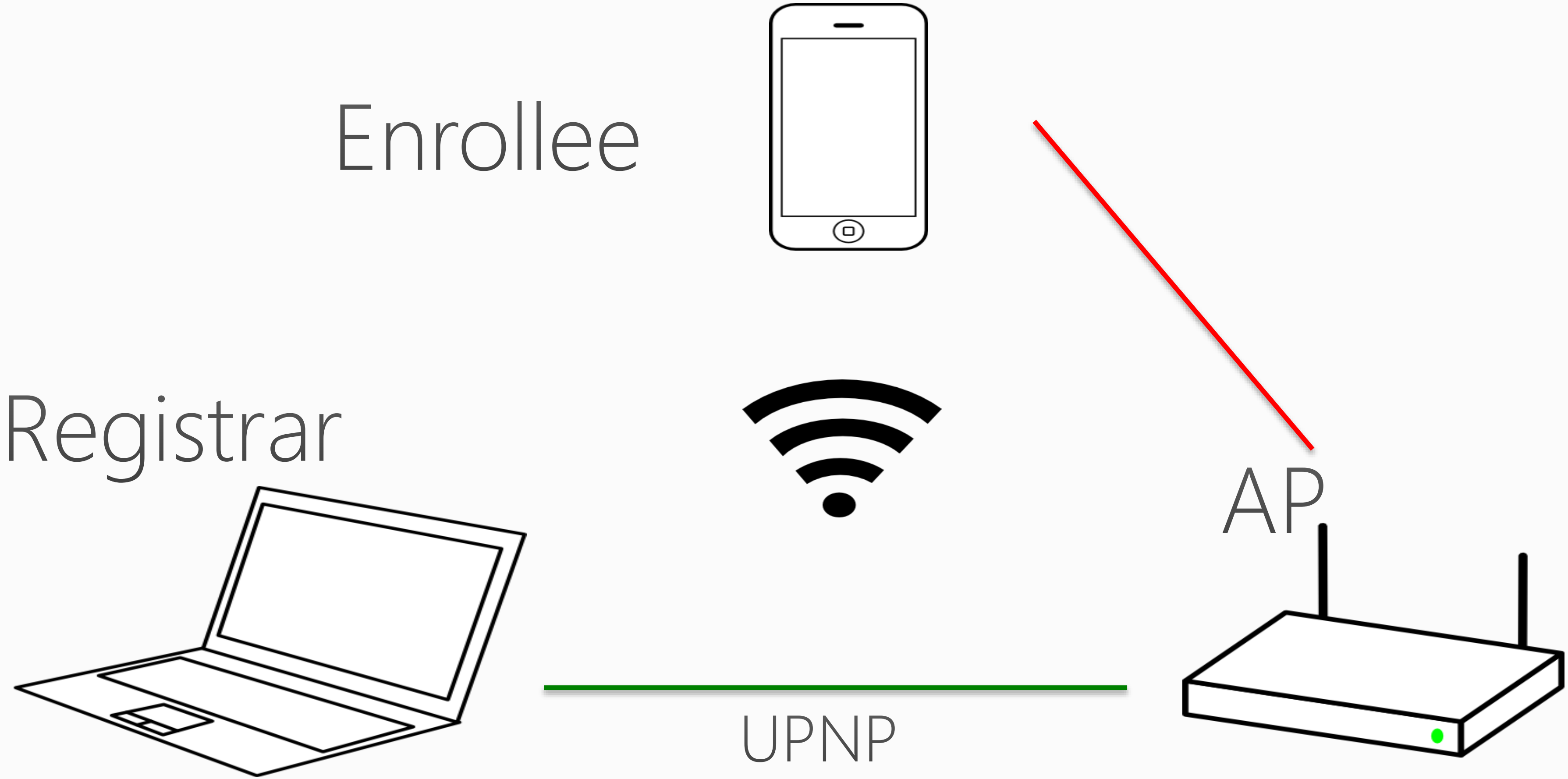
# WPS Components



# WPS Components



# WPS Components



# WPS Components (usual setup)

# WPS Components

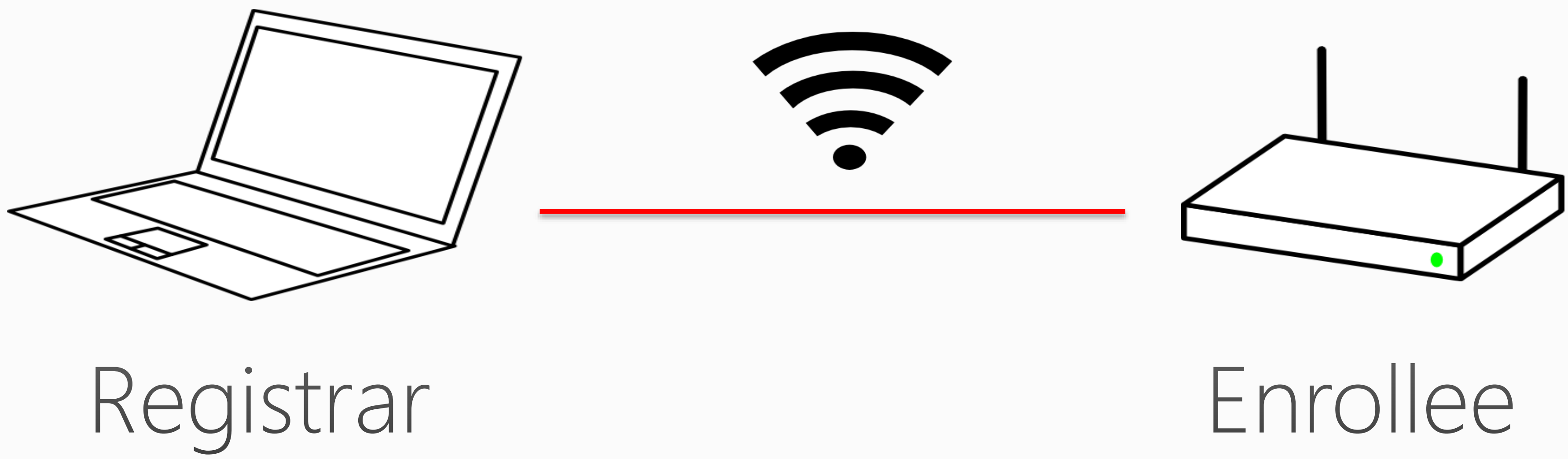
- An Enrollee can be a station or an AP
- A Registrar can be a station (external registrar) or an AP
- A Registrar doesn‘t need to be in the WiFi network
- A  WiFi network can have more than one WPS Registrar

# WPS Misconception 2

- In the most common case, the **Registrar** is a station outside the WiFi network and the **Enrollee** is the AP, not the other way around.

# WPS PIN External Registrar Protocol

# WPS PIN External Registrar Architecture



# WPS PIN formats

## User selected PIN

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |

## PIN printed on sticker

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 1 | 2 | 3 | 4 | 5 | 6 | 7 | X |

Checksum

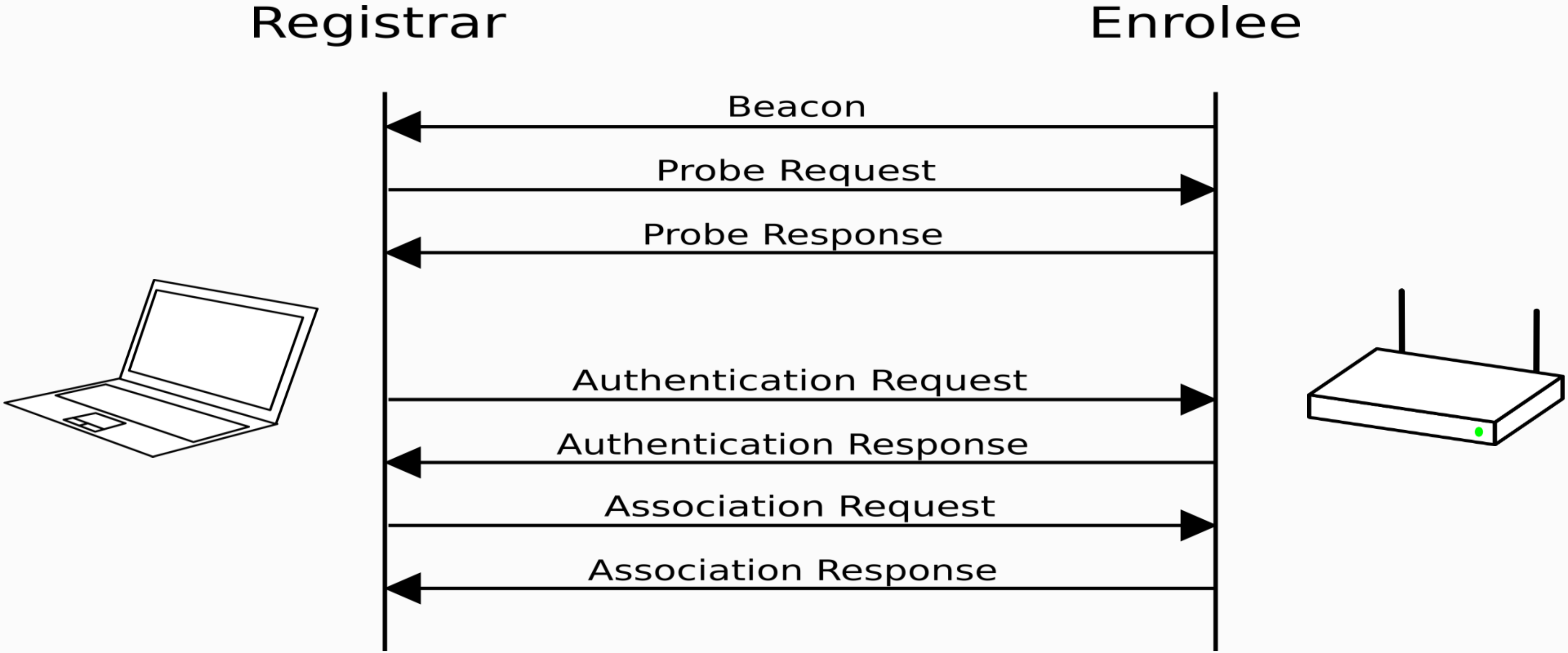
# Quotes from the Spec. (1.0h- 2006)

The recommended length for a manually entered device password is an 8-digit numeric PIN. This length does not provide a large amount of entropy for strong mutual authentication, but the design of the Registration Protocol protects against dictionary attacks on PINs **if a fresh PIN or a rekeying key is used each time the Registration Protocol is run.**

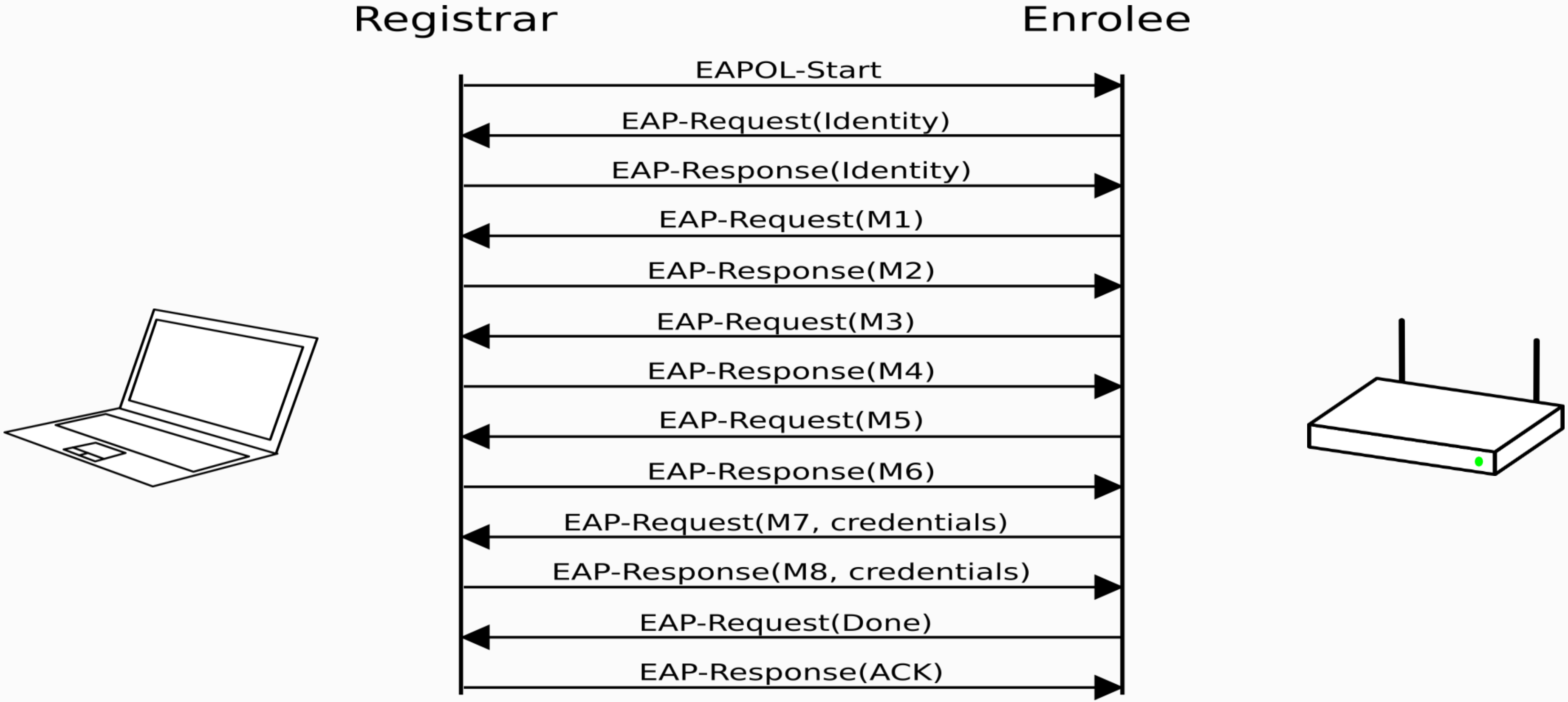
If the Registrar runs the Protocol multiple times using the same PIN an attacker will be able to discover the PIN through brute force. **To address this vulnerability, if a PIN authentication error occurs, the Registrar SHALL warn the user and SHALL NOT automatically reuse the PIN.**

**The [sticker] PIN contains approximately 23 bits of entropy**… It is susceptible to active attack.

# WPS PIN External Registrar protocol



# WPS PIN External Registrar protocol



# WPS PIN External Registrar protocol

Pre-commitment


Provides mutual-authentication

# The M1 message

| | | |
|---|---|---|
| **E -> R** | **M1** | **N1 \|\| Description \|\| PKE** |

- **N1** is a 128-bit random nonce generated by the Enrollee
- **PKE** is the DH public key of the Enrollee

# Key derivation

- Upon reception of M1 the Registrar generates PKR and N2
- The Registrar can then compute the DHKey:

$$\text{DHKey} = \text{SHA-256}(\text{zeropad}(g^{AB} \bmod p, 192))$$

- And calculate the Key Derivation Key :

$$\text{KDK} = \text{HMAC-SHA-256DHKey}(\text{N1} \,||\, \text{EnrolleeMAC} \,||\, \text{N2})$$

- Finally AuthKey, KeyWrapKey, and EMSK are derived:

$$\text{AuthKey} \,||\, \text{KeyWrapKey} \,||\, \text{EMSK} =$$

$$\text{kdf}(\text{KDK}, \text{“Wi-Fi Easy and Secure Key Derivation”}, 640)$$

# Key derivation

- **AuthKey** : used to authenticate the Registration Protocol messages (256 bits)
- **KeyWrapKey** : used to encrypt secret nonces and ConfigData (128 bits)
- **EMSK** : Extended Master Session Key that is used to derive additional keys (256 bits)

# The M2 message

| | | |
|---|---|---|
| **R -> E** | **M2** | **N1 \|\| N2 \|\| Desc. \|\| PKR \|\| Auth** |

- **N2** is a 128-bit random nonce generated by the Registrar
- **PKR** is the DH public key of the Registrar
- **Auth** = $\mathrm{HMAC}_{\mathrm{AuthKey}}(M1 \mid\mid M2)$

# The M3 message

| E -> R | M3 | E-Hash1 || E-Hash2 |
|---|---|---|

- **E-Hash1** = $HMAC_{AuthKey}(E\text{-}S1 \,||\, PSK1 \,||\, PKE \,||\, PKR)$
- **E-Hash2** = $HMAC_{AuthKey}(E\text{-}S2 \,||\, PSK2 \,||\, PKE \,||\, PKR)$
- **PSK1** is made of the first 4 digits of the PIN
- **PSK2** is made of the last 4 digits of the PIN
- **E-S1** and **E-S2** are two 128 bit random nonces

# The M4 message

| | | |
|---|---|---|
| **R -> E** | **M4** | **R-Hash1 \|\| R-Hash2 \|\| $E_{Kwk}$(R-S1)** |

- **R-Hash1** = $HMAC_{AuthKey}$(R-S1 || PSK1 || PKE || PKR)
- **R-Hash2** = $HMAC_{AuthKey}$(R-S2 || PSK2 || PKE || PKR)
- **R-S1** and **R-S2** are two 128 bit random nonces

# M4 message processsing

- The Enrollee decrypts R-S1
- The Enrollee verifies :

$$HMAC_{AuthKey}(R\text{-}S1 \,||\, PSK1 \,||\, PKE \,||\, PKR) = R\text{-}Hash1$$

# The M5 message

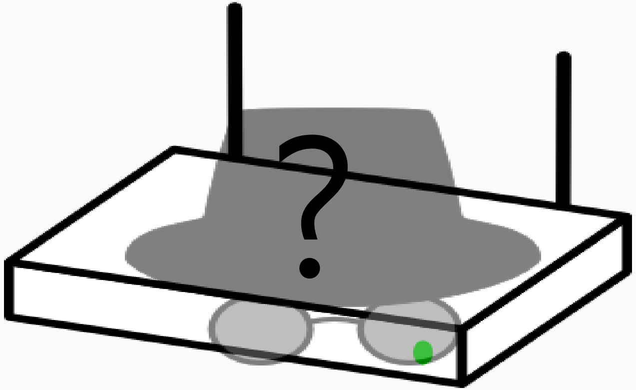
| | | |
|---|---|---|
| E -> R | M5 | $E_{kwk}(E\text{-}S1)$ |

- The Enrollee opens its first commitment

# M5 message processsing

- The Registrar decrypts E-S1
- The Registrar verifies :

$$HMAC_{AuthKey}(E\text{-}S1 \,||\, PSK1 \,||\, PKE \,||\, PKR) = E\text{-}Hash1$$



# The M6 message

| | | |
|---|---|---|
| **R -> E** | **M6** | $\mathbf{E_{Kwk}(R\text{-}S2)}$ |

- The registrar opens its second commitment

$$\mathbf{HMAC_{AuthKey}(R\text{-}S2 \,||\, PSK2 \,||\, PKE \,||\, PKR) = E\text{-}Hash2\ ?}$$

# The M7 message

| | | |
|---|---|---|
| **E -> R** | **M7** | $E_{kwk}(E\text{-}S2 \,\|\, \text{Credentials})$ |

- The Enrollee opens its second commitment and also sends the network credentials

# WPS AP as Registrar attack

# WPS PIN External Registrar protocol

- Why is the AP the Registrar resp. the Station the Enrollee  and not the other way around?
- The WiFi Alliance probably found out that the protocol would otherwise be totally insecure in the scenario with Headless devices

# The M1 message (AP as Registrar)

**E -> R** **M1** **N1 || Description || PKE**

- **N1** is a 128-bit random nonce generated by the Enrollee
- **PKE** is the DH public key of the Enrollee

# The M2 message (AP as Registrar)

| | | |
|---|---|---|
| **R -> E** | **M2** | **N1 \|\| N2 \|\| Desc. \|\| PKR \|\| Auth** |

- **N2** is a 128-bit random nonce generated by the Registrar
- **PKR** is the DH public key of the Registrar
- **Auth** = $\mathrm{HMAC}_{\mathrm{AuthKey}}(M1 \,||\, M2)$

# The M3 message (AP as Registrar)

| E -> R | M3 | E-Hash1 || E-Hash2 |
|---|---|---|

- **E-Hash1 = Random**
- **E-Hash2 = Random**

# The M4 message (AP as Registrar)

| | | |
|---|---|---|
| R -> E | M4 | R-Hash1 \|\| R-Hash2 \|\| $E_{Kwk}$(R-S1) |

- **The Enrollee can decrypt R-S1 and then brute force PSK1 with R-Hash1**
- **The Enrollee then restarts the protocol knowing PSK1**

# The M5 message (AP as Registrar)

| E -> R | M5 | $E_{kwk}$(E-S1) |
| --- | --- | --- |

- In the second run of the protocol, the Enrollee can send valid values since it knows PSK1

# The M6 message (AP as Registrar)

| R -> E | M6 | $E_{Kwk}$(R-S2) |
|---|---|---|

- **The Enrollee can decrypt R-S2 and then brute force PSK2 with R-Hash2**
- **The Enrollee then restarts the protocol one last time knowing both PSK1 and PSK2**

# WPS online bruteforce attack

# WPS PIN External Registrar protocol

- Looks **OK as long as there is only one try per PIN**
- Proof of possession allows detection of rogue APs and stations
- The DH key exchange protects against eavesdropping

# WPS online BF attack

## .braindump – RE and stuff

```
; Attributes: bp-based frame

follow_me proc near
push    rbp
mov     rbp, rsp
sub     rsp, 20h
lea     rcx, a@sviehb   ; "@sviehb"
call    printf
add     rsp, 20h
pop     rbp
retn
follow_me endp
```

**December 27, 2011**

### Wi-Fi Protected Setup PIN brute force vulnerability

Filed under: advisories — Stefan @ 3:00 am

A few weeks ago I decided to take a look at the Wi-Fi Protected Setup (WPS) technology. I noticed a few really bad design decisions which enable an efficient brute force attack, thus effectively breaking the security of pretty much all WPS-enabled Wi-Fi routers. As all of the more recent router models come with WPS enabled by default, this affects millions of devices worldwide.

I reported this vulnerability to CERT/CC and provided them with a list of (confirmed) affected vendors. CERT/CC has assigned VU#723755 to this issue.
To my knowledge **none** of the vendors have reacted and released firmware with mitigations in place.

Detailed information about this vulnerability can be found in this paper: **Brute forcing Wi-Fi Protected Setup** – Please keep in mind that the devices mentioned there are just a tiny subset of the affected devices.

I would like to thank the guys at CERT for coordinating this vulnerability.

**Update (12/29/2011 – 20:15 CET)**
As you probably already know, this vulnerability was **independently** discovered by Craig Heffner (/dev/ttyS0, Tactical Network Solutions) as well – I was just the one who reported the vulnerability and released information about it first. Craig and his team have now released their tool "Reaver" over at Google Code.

My PoC Brute Force Tool can be found here. It's a bit faster than Reaver, but will not work with all Wi-Fi adapters.

# WPS online brute force attack

- Attack published in 2011 by Stefan Viehböck
- The idea is to bruteforce PSK1 and then PSK2
- Takes at most 11’000 trials for sticker PIN
  - At most 20’000 trials for user selected PIN
- Finds the PIN in a few hours (depends on AP)
- Most AP implemented no security against BF
- **Implemented in tools like Reaver and Bully**

# Countermeasures

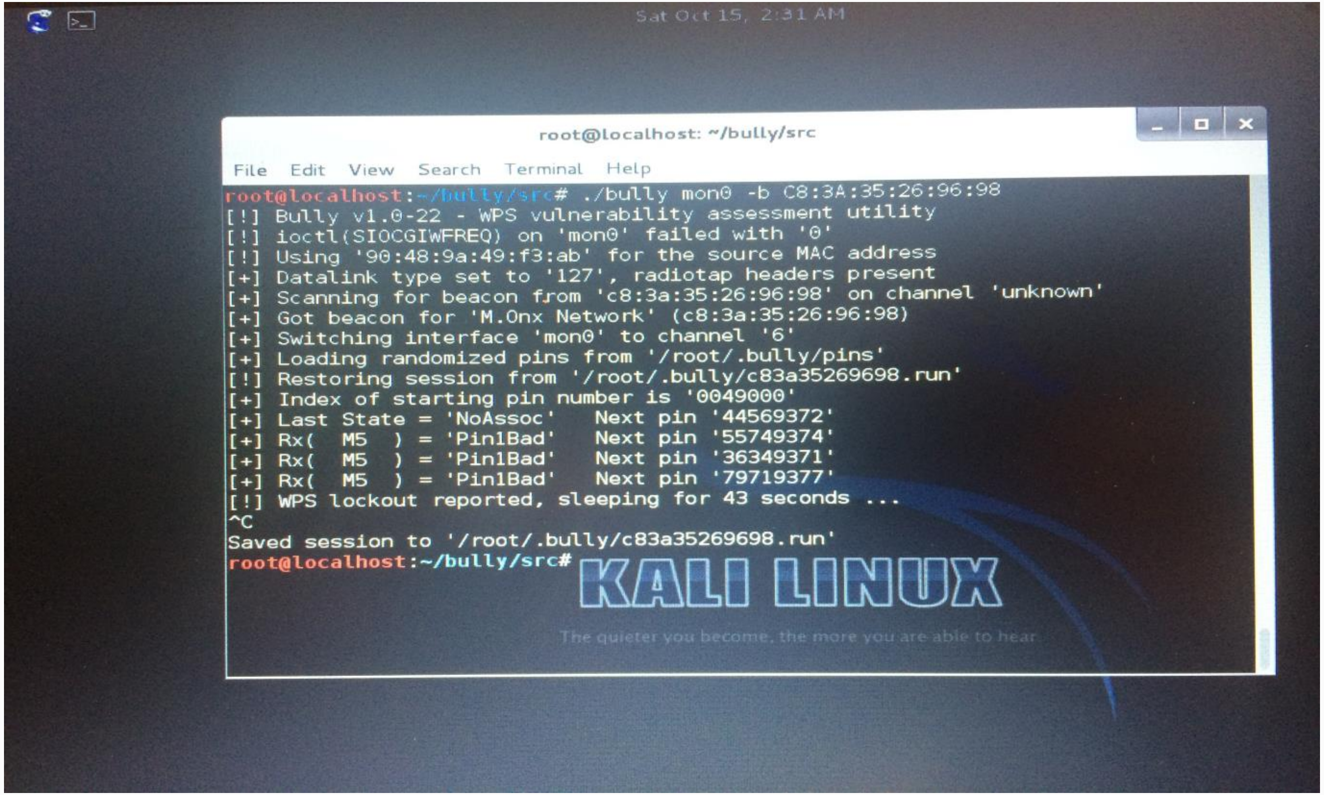
- Changes in the specification

2.0.2 Public release version

- Change Headless Devices section to mandate implementation of strong mitigation against a brute force attack on the AP that uses a static PIN.

- Some devices have a WPS lockout delay
  - This only slows down the attack a bit
- Other lock WPS until the next reboot

# WPS lock-out shown in Bully



# Counter-Countermeasures

- AP reboot scripts (mdk3, ReVdK3)
  - EAPOL-Start flood attack
- Deauth DDoS

# Conjecture

- The initial use case seems to be random PIN on display with one try
- The specification contains contradictory statements about PIN reuse
- The protocol looks secure enough if PINs are not reused

## Conclusion:

- Headless devices with static PINs were probably a last minute addition to the specification

# WPS offline bruteforce attack

# The M1 message

| | | |
|---|---|---|
| **E -> R** | **M1** | **N1 \|\| Description \|\| PKE** |

- **N1** is a 128-bit random nonce generated by the Enrollee
- **PKE** is the DH public key of the Enrollee

# The M3 message

| | | |
|---|---|---|
| **E -> R** | **M3** | **E-Hash1 \|\| E-Hash2** |

- **E-Hash1** = $HMAC_{AuthKey}(E\text{-}S1 \,\|\, PSK1 \,\|\, PKE \,\|\, PKR)$
- **E-Hash2** = $HMAC_{AuthKey}(E\text{-}S2 \,\|\, PSK2 \,\|\, PKE \,\|\, PKR)$
- **PSK1** is made of the first 4 digits of the PIN
- **PSK2** is made of the last 4 digits of the PIN

# The offline attack

- If we can find E-S1 and E-S2, we can the brute force PSK1 and PSK2 offline!

# How are E-S1 and E-S2 generated?

- Usually with pseudo-random generators (PRNG)
- **Often insecure PRNG**
  - No or low entropy
  - Small state (32 bits)
- Can the PRNG state be recovered ?

# Implementation in Broadcom/eCos

```
reg_proto_create_m1(RegData *regInfo, BufferObj *msg)
{
	uint32 ret = WPS_SUCCESS;
	uint8 message;

	DevInfo *enrollee = regInfo->enrollee;

	/* First generate/gather all the required data. */
	message = WPS_ID_MESSAGE_M1;

	/* Enrollee nonce */
	/*
	 * Hacking, do not generate new random enrollee nonce
	 * in case of we have prebuild enrollee nonce.
	 */
	if (regInfo->e_lastMsgSent == MNONE) {
		RAND_bytes(regInfo->enrolleeNonce, SIZE_128_BITS);
	}
	/* It should not generate new key pair if we have prebuild enrollee nonce */
	if (!enrollee->DHSecret) {
		ret = reg_proto_generate_dhkeypair(&enrollee->DHSecret);
		if (ret != WPS_SUCCESS) {
			return ret;
		}
	}
	...
```

https://raw.githubusercontent.com/RMerl/asuswrt-merlin/master/release/src-rt-6.x/bcmcrypto/random.c

# Implementation in Broadcom/eCos

```
#if (defined(__ECOS) || defined(TARGETOS_nucleus) || defined(TARGETOS_symbian))

void generic_random(uint8 * random, int len)
{
    int tlen = len;
    while (tlen--) {
        *random = (uint8)rand();
        *random++;
    }
    return;
}

#endif
```

https://raw.githubusercontent.com/RMerl/asuswrt-merlin/master/release/src-rt-6.x/bcmcrypto/random.c

# Implementation in Broadcom/eCos

```
int rand_r( unsigned int *seed )  {

    unsigned int s=*seed;
    unsigned int uret;
    s = (s * 1103515245) + 12345; // permutate seed
    uret = s & 0xffe00000;  // Only use top 11 bits

    s = (s * 1103515245) + 12345; // permutate seed
    uret += (s & 0xfffc0000) >> 11;    // Only use top 14 bits

    s = (s * 1103515245) + 12345; // permutate seed
    uret += (s & 0xfe000000) >> (11+14);  // Only use top 7 bits

    retval = (int)(uret & RAND_MAX);
    *seed = s;
    return retval;
}
```

http://trac.umnaem.webfactional.com/browser/trunk/Hardware/eCos/packages/language/c/libc/stdlib/v3_0/src/rand.cxx?rev=39

# Implementation in Broadcom/eCos

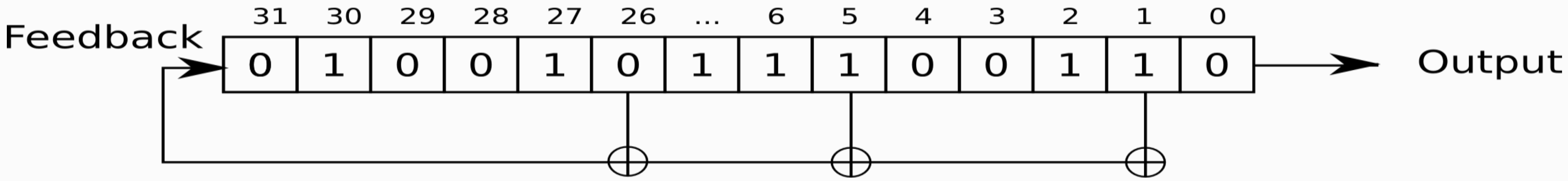
- Linear Congruential Generator
  - 32 bits state
  - No external entropy
  - **E-S1** and **E-S2** generated right after **N1**
  - Optimization: 7 bits of the seed can be deduced from the last output byte

# The attack in details

- Do the WPS protocol up to message M3
- Get the Nonce from M1
  - Bruteforce the state of the PRNG
- Compute E-S1 and E-S2 from the state
- Bruteforce PSK1 / PSK2 from E-Hash1 / E-Hash2
- Do the full WPS protocol to get the credentials

# Random implementation in Ralink

- 32 bit Linear Feedback Shift Register (LFSR)




- Polynomial = 0x80000057
- Trivial to recover the LFSR state from the nonce

# Random implementation in Ralink

- E-S1 and E-S2 are never generated
- E-S1 = E-S2 = 0x0

# Low entropy at boot

- Some AP have the same state at each boot
- Make a list of common states after reboot
- Attack the AP right after boot
- As shown, there are many ways to force a reboot

# Linux / Hostapd

- Looks okay
- Uses /dev/random
- **Used in Atheros SDK**
- But you never know
  - Several papers attack the entropy of the linux PRNG in embedded systems

# Manufactures not yet checked

- Marvell
- Realtek
- Intel
- Qualcomm
- …

# How prevalent is the problem?

- It's complicated
- Many of the implementations are the reference code for the chipset
  - Only the GUI is reskinned
  - Therefore many brands are affected
- Many vendors use different chipset

# Vendor responses

# Broadcom

- Tried to find a security incident contact
- Tried to contact them on Twitter
- Tried to contact them through their website

# Broadcom

Dominique Bongard discovered that Broadcom chips are affected. Their random number generators apparently are so easy to guess that an attacker can get your Wi-Fi access point to give up its PIN code in less than a second.

-------------------------------------

This is the first we have heard of this. We'll connect with your security team.
Karen

# Broadcom

Thanks for checking. This is not a chip issue. The issue you have identified can affect any Wi-Fi product.

Vulnerabilities can depend on the Wi-Fi standard that is chosen for security. This may depend on the age of the product.

Best regards,

Jennifer B.| Senior Manager, Corporate Communications

# Cisco

We do use the Broadcom chipset in some of our offerings, and we're reaching out to Broadcom as we speak, to find out if any of the ones we use are affected by this issue.

[...] Also, for your information - Cisco has a very limited number of wireless products with support for WPS. Most of them are Small and Medium business products, while others are sold to Service Providers (not to end users) to be used as cable modem CPEs. And some of those CPEs have wireless capabilities, and some support WPS. We'll investigate them all, make our results public by following our security policy.

# Ralink

- Tried to contact them via their website

# WiFi Alliance

Thanks, Dominique. This is very helpful.

In the future, I encourage you to report any Wi-Fi-related vulnerabilities directly to us. Wi-Fi Alliance reviews all submitted reports of security vulnerabilities affecting Wi-Fi CERTIFIED programs. You can submit vulnerabilities to secure@wi-fi.org or at https://www.wi-fi.org/secure .

Thanks again.

Regards,

Kevin R. | Director of Program Marketing | Wi-Fi Alliance

# WPS static pin generation attack

# More quotes from the specification

PIN values should be randomly generated, and they SHALL NOT be derivable from any information that can be obtained by an eavesdropper or active attacker. The device’s serial number and MAC address, for example, are easily eavesdropped by an attacker on the in-band channel.

# Difference between theory and practice

| | |
|---|---|
| Arris | http://packetstormsecurity.com/files/123631/ARRIS-DG860A-WPS-PIN-Generator.html |
| Belkin | http://ednolo.alumnos.upv.es/?p=1295 |
| Other | http://www.hackforums.net/printthread.php?tid=4146055 |
| … * | Tenda, Sitecom, Linksys, FTE, Vodafone, ZTE, Zyxel |

* WPSPIN :http://www.crack-wifi.com/forum/topic-8793-wpspin-generateur-pin-wps-par-defaut-routeurs-huawei-belkin.html

# Conclusion

# Conclusion

- Disable WPS now !
- Reverse engineers: Check other AP for bad PRNG
- Cryptographers: Check if good PRNG are okay