HACK.LU 2007
18-20 October
Kirchberg-Luxembourg
http://www.hack.lu/

If you want to participate :
Call for Paper, Call for Poster,
Lightning Talk and more...

Wi-Fi Implementation Bugs: an Era of New Vulnerabilities

Laurent BUTTI – Julien TINNES – Franck VEYSSET
France Télécom R&D / Orange Labs
firstname dot lastname at orange-ftgroup dot com

orange™

research & development

WhoAreWe mandatory slide

- Network security experts in R&D labs
  - Working for France Telecom – Orange (a major telco)
- Speakers at security-focused conferences
  - SSTIC, BlackHat US & Europe, ToorCon, ShmooCon, FIRST, hack.lu …
- Wi-Fi security centric ;-)
  - “Wi-Fi Security: What’s Next” – ToorCon 2003
  - “Design and Implementation of a Wireless IDS” – ToorCon 2004 and ShmooCon 2005
  - “Wi-Fi Trickery, or How To Secure (?), Break (??) and Have Fun With Wi-Fi” – ShmooCon 2006
  - “Wi-Fi Advanced Stealth” – BlackHat US 2006 and Hack.LU 2006
  - “Wi-Fi Advanced Fuzzing” – BlackHat EU 2007

Agenda

- Forewords
- Finding 802.11 implementation bugs: 802.11 fuzzing
- Client side implementation bugs
  - Details on one of the four client-related vulnerabilities we discovered
  - The first “public” Linux-based kernel remote exploitation, due to a 802.11 driver implementation flaw
- Wireless access point vulnerabilities
  - Disclosure of a new vulnerability today

Goals of This Talk

- Provide the audience with
  - A quick overview of current 802.11 fuzzing techniques that allowed us to discover several critical implementation bugs
  - Some recent research on access point fuzzing with interesting findings
  - A description of the *first* remote linux-based kernel exploit on 802.11 that is now integrated in Metasploit
  - Some live demos!

Forewords

Facts

- Wi-Fi weakens entreprise's perimetric security
  - Weak Wi-Fi network infrastructures (open, WEP, misconfigured WPA)
  - Rogue or misconfigured access points (open access points)
- But also weakens client's security
  - Rogue access points in public zones (conferences, hot spots…)
  - Fake access points attacking (automagically) clients (KARMA)
  - Traffic injection within clients' communications (AIRPWN, WIFITAP)
- Unfortunately all these issues are hardly detectable
  - Without specific tools (Wireless IDS…)
- But wait… There is more to come…

What We Guessed…

- Implementation bugs in 802.11 drivers
  - Developed in C/C++
  - Numerous chipsets ⇨ Numerous developers ⇨ Heterogeneous implementations regarding security

- Promising implementation bugs!
  - Potential arbitrary kernel-mode code execution
    - Bypassing all classic security mechanisms: AV, PFW, HIPS…
  - Remotely triggerable within the victim's radio coverage
    - Not necessarly been associated to a rogue access point!
  - Even if security mechanisms are activated (WPA/WPA2)

What Happened…

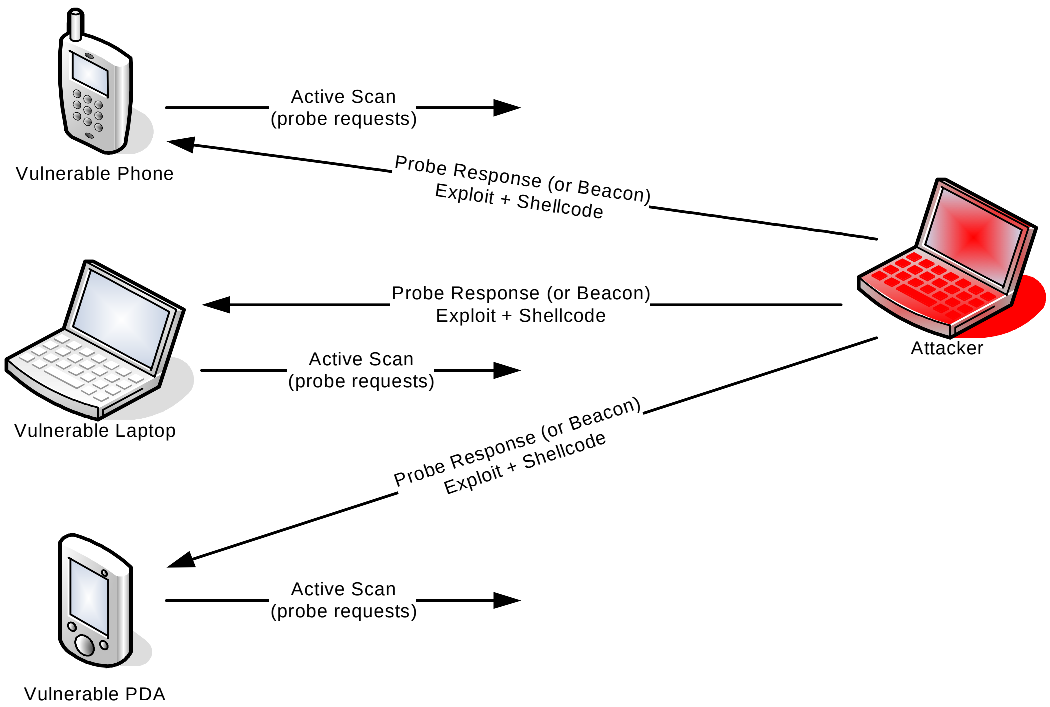
- First public announcement at BlackHat US 2006
  - Johnny Cache and David Maynor presentation [DEVICEDRIVERS]
- Month of Kernel Bugs on November, 2006 [MOKB]
  - Apple Airport 802.11 Probe Response Kernel Memory Corruption (OS X)
  - Broadcom Wireless Driver Probe Response SSID Overflow (Windows)
  - D-Link DWL-G132 Wireless Driver Beacon Rates Overflow (Windows)
  - NetGear WG111v2 Wireless Driver Long Beacon Overflow (Windows)
  - NetGear MA521 Wireless Driver Long Rates Overflow (Windows) (*)
  - NetGear WG311v1 Wireless Driver Long SSID Overflow (Windows) (*)
  - Apple Airport Extreme Beacon Frame Denial of Service (OS X)
- But also under Linux
  - Madwifi stack-based overflow (*)
    - Potentially all recent unpatched Linux distributions running on an Atheros chipset

(*) found by our fuzzer

Potential Targets?

- Nowadays Wi-Fi technologies are ubiquitous!
  - All recent laptops
  - Most entreprises are equipped with Wi-Fi devices
  - More and more home boxes (DSL gateways…)
  - More and more cellular phones (VoIPoWLAN)
  - Video gaming consoles, digital cameras, printers…
- But also, protection / analyser mechanisms may be vulnerable
  - e.g. wireless IDS/IPS, sniffers (tcpdump, wireshark)…
- So many (potentially) vulnerable 802.11 implementations!

802.11 Station Attack Overview



- 802.11 exploits a.k.a. 0wn3d by a 802.11 frame!

1^{st} Step: Finding These Vulnerabilities!

- Closed source drivers
  - Black box testing
  - Reverse engineering
- Open source drivers
  - Black / White box testing
  - Source code auditing
- Reverse engineering drivers is time consuming
  - Especially when you haven't any clue…
- Black box testing may be useful in both cases…

Fuzzing 101

research & development

Fuzzing? (1/2)

- Really hard to define…
  - Security community / industry love this kind of hyped / buzzed words! ;-)

- Some definitions
  - Fuzz Testing or Fuzzing is a Black Box software testing technique, which basically consists in finding implementation bugs using malformed or semi malformed data injection in a automated fashion. [OWASP]
  - Fuzz testing or fuzzing is a software testing technique. The basic idea is to attach the inputs of a program to a source of random data ("fuzz"). If the program fails (for example, by crashing, or by failing built-in code assertions), then there are defects to correct. [WIKIPEDIA]

- Common part
  - Software testing technique that consists in finding implementation bugs
    - 1st definition: with malformed or semi malformed data injection
    - 2nd definition: with random data

Fuzzing? (2/2)

- Fuzzing is by far one of the best price / earning ratio ;-)
  - Reverse engineering load of drivers is costly and boring
  - Implementing a basic fuzzer may be low cost
  - Discovered implementation bugs will thus be the most obvious ones
- But fuzzing will (probably) not help you finding “complex” bugs
  - Simply because all testing possibilities cannot be performed due to
    - Lack of time versus all test possibilities
    - Protocol specificities (states)
- Of course, investigations on exploitation requires reverse engineering and/or source code auditing

Some Fuzzing Successes

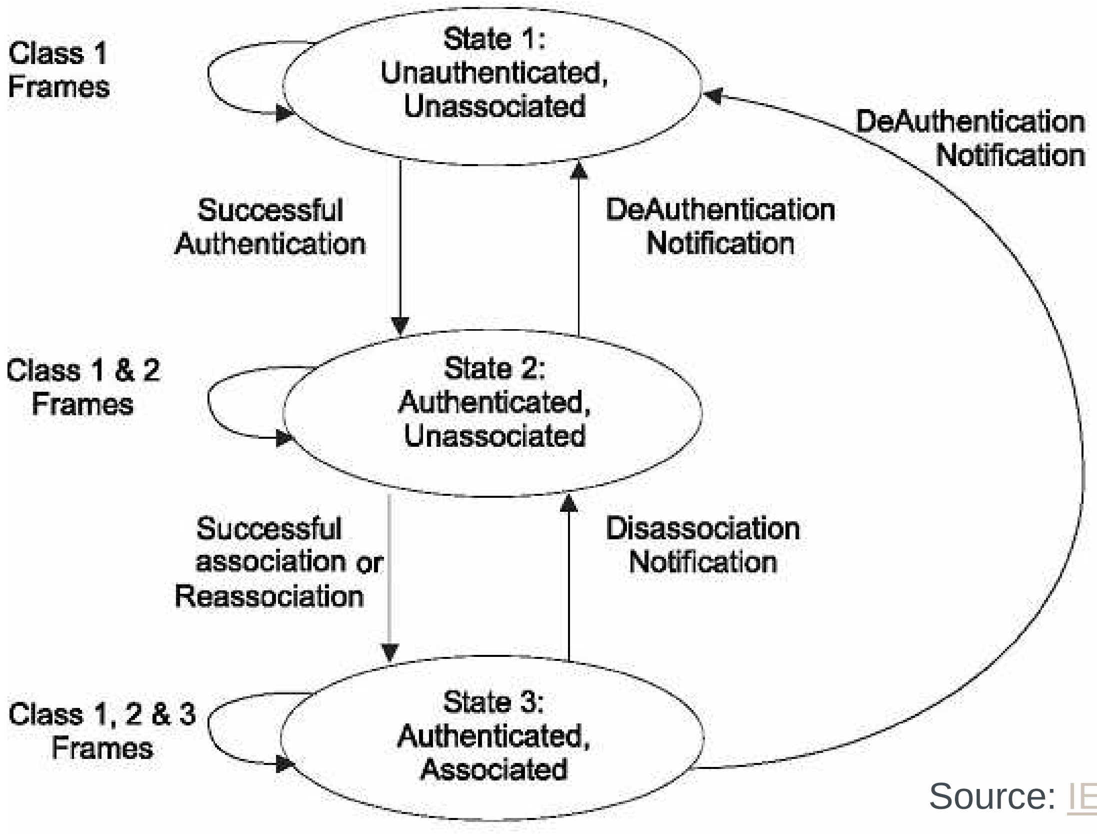
- Month of “Whatever” Bugs
  - Most vulnerabilities discovered thanks to fuzzing techniques
- Take a look at LMH’s **`fsfuzzer`**
  - Really basic but _so_ effective!
- Some open source fuzzers
  - SPIKE (Immunity): multi-purpose fuzzer
  - PROTOS suite (Oulu University): SIP, SNMP…
  - Sulley Fuzzing Framework

Fuzzing 802.11 Stacks

802.11 Fuzzing? (1/2)

- 802.11 legacy standard is somewhat complex
  - Several frame types (management, data, control)
  - Lot of signalling
    - Rates, channel, network name, cryptographic capabilities, proprietary capabilities…
  - All this stuff must be parsed by the firmware/driver!
- 802.11 extensions are more and more complex!
  - 802.11i for security, 802.11e for QoS…
  - 802.11w, 802.11r, 802.11k…
- Complexity++ ⇨ Code++ ⇨ Bugs++

802.11 Fuzzing? (2/3)



Source: IEEE 802.11-1999

- 802.11 states are fuzzable
  - State 1: initial start, unauthenticated, unassociated (e.g. scanning process)
  - State 2: authenticated, unassociated
  - State 3: authenticated, associated

802.11 Fuzzing? (3/3)

- Scanning procedure of 802.11 client stacks can be fuzzed
  - Active scanning: send probe requests and listen to probe responses back, and do channel hopping
  - Passive scanning: listen to beacons and do channel hopping
  - Note: drivers may be listening to both beacons and probe responses

802.11 Overview 101

- MAC frame format

Octets: 2	2	6	6	6	2	6	0 - 2312	4
Frame Control	Duration/ ID	Address 1	Address 2	Address 3	Sequence Control	Address 4	Frame Body	FCS

MAC Header (Frame Control through Address 4)

- Frame Control defines upper layer (frame body)

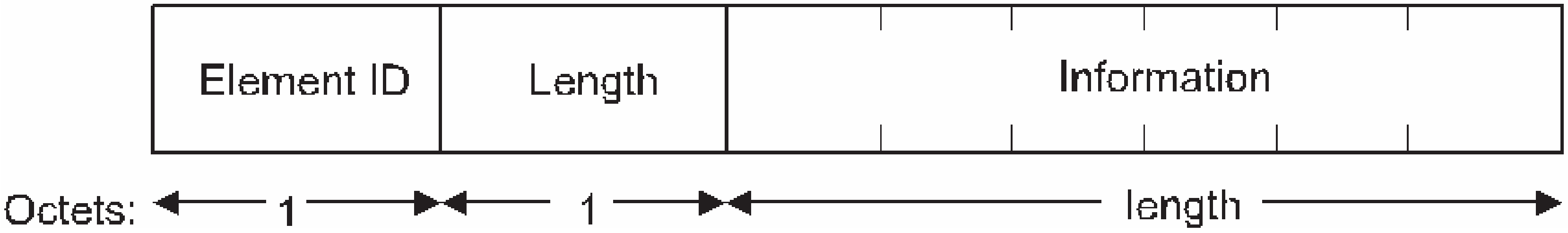
B0 B1	B2 B3	B4 B7	B8	B9	B10	B11	B12	B13	B14	B15
Protocol Version	Type	Subtype	To DS	From DS	More Frag	Retry	Pwr Mgt	More Data	WEP	Order
Bits: 2	2	4	1	1	1	1	1	1	1	1

802.11 Overview 101

- Beacon / Probe Response format

Order	Information	Notes
1	Timestamp	
2	Beacon interval	
3	Capability information	
4	SSID	
5	Supported rates	
6	FH Parameter Set	The FH Parameter Set information element is present within Beacon frames generated by STAs using frequency-hopping PHYs.
7	DS Parameter Set	The DS Parameter Set information element is present within Beacon frames generated by STAs using direct sequence PHYs.
8	CF Parameter Set	The CF Parameter Set information element is only present within Beacon frames generated by APs supporting a PCF.
9	IBSS Parameter Set	The IBSS Parameter Set information element is only present within Beacon frames generated by STAs in an IBSS.
10	TIM	The TIM information element is only present within Beacon frames generated by APs.

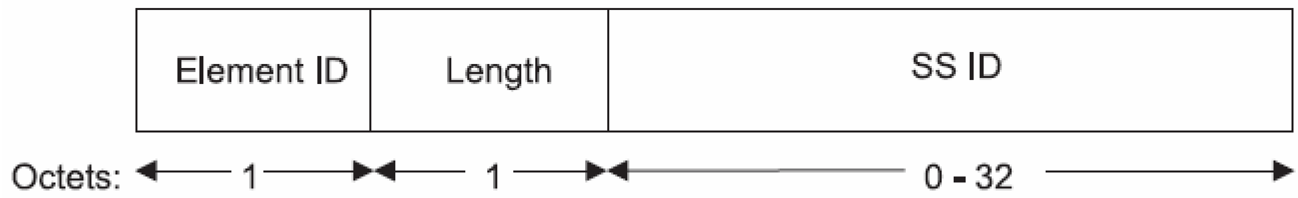
802.11 Overview 101

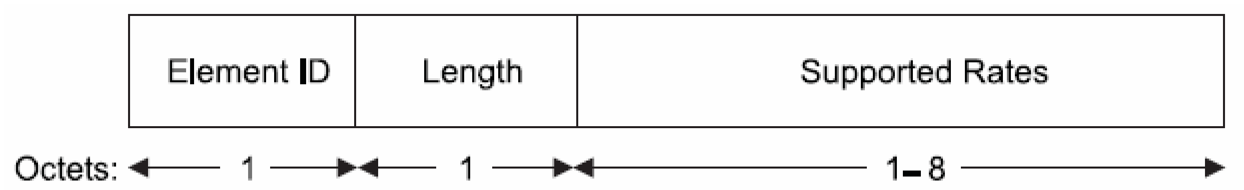


Information element	Element ID
SSID	0
Supported rates	1
FH Parameter Set	2
DS Parameter Set	3
CF Parameter Set	4
TIM	5
IBSS Parameter Set	6
Reserved	7–15
Challenge text	16
Reserved for challenge text extension	17–31
Reserved	32–255

802.11 Overview 101

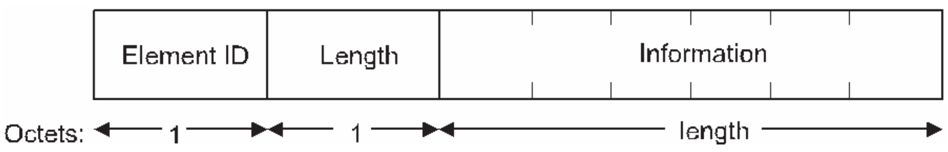
- Some Information Elements

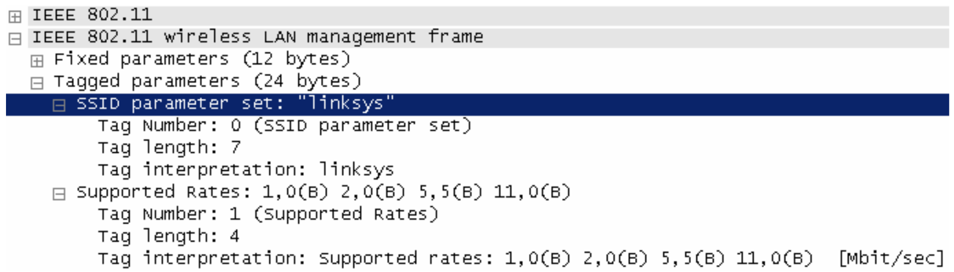




Fuzzing Information Elements

- A (good) candidate for 802.11 fuzzing: the Information Element
  - Type / Length / Value
  - Type is the Element ID (1 byte)
  - Length is the total length of the Value payload (1 byte)
  - Value is the payload of the Information Element (0-255 bytes)





- Most IEs have a fixed or maximum length
  - Take the length within 802.11 frame
  - If unproperly checked: possible overflows

Check This for More Information

- “Wi-Fi Advanced Fuzzing” – Black Hat EU 2007
  - https://www.blackhat.com/presentations/bh-europe-07/Butti/Presentation/bh-eu-07-Butti.pdf

Discovered Vulnerabilities (Client Implementations)

Discovered Vulnerabilties

- NetGear MA521 Wireless Driver Long Rates Overflow (CVE-2006-6059)
  - Overflowing Rates Information Element
    - This field has generally a maximum length of 8 bytes (implementation dependent)
- NetGear WG311v1 Wireless Driver Long SSID Overflow (CVE-2006-6125)
  - Overflowing SSID Information Element
    - This field has a maximum length of 32 bytes
- D-Link DWL-G650+ (A1) Wireless Driver Long TIM Overflow (CVE-2007-0993)
  - Overflowing TIM Information Element
- Madwifi Driver Remote Buffer Overflow Vulnerability (CVE-2006-6332)
  - Overflowing WPA/RSN/WMM/ATH Information Element
  - Triggered when SIOCGIWSCAN
    - e.g. thanks to **`iwlist`** or **`iwlib.h`**

Exploitation

The Madwifi flaw

- By using the fuzzer, we get an OOPS
  - Registers states
  - Stack state
  - Backtrace
- We almost immediately notice the value of EBP and EIP
- The backtrace shows us that we're in some ioctl() system call
  - This means process context
  - But kernel mode!

The flaw

- We can quickly conclude to a kernel stack buffer overflow
  - We can find the vulnerable function by using the backtrace: (giwscan_cb)
  - char buf[64 * 2 + 30];
  - memcpy(buf, se->se_wpa_ie, se->se_wpa_ie[1] + 2);
  - We control the size in memcpy. Ouch!
- It is possible to craft a very malicious 802.11 frame

Consequences

- We've reported this flaw, with a patch, in december 2006
- Madwifi published a new fixed version the following day
- Linux distributions could begin to patch update their madwifi drivers
  - Unfortunately some did'nt react quickly

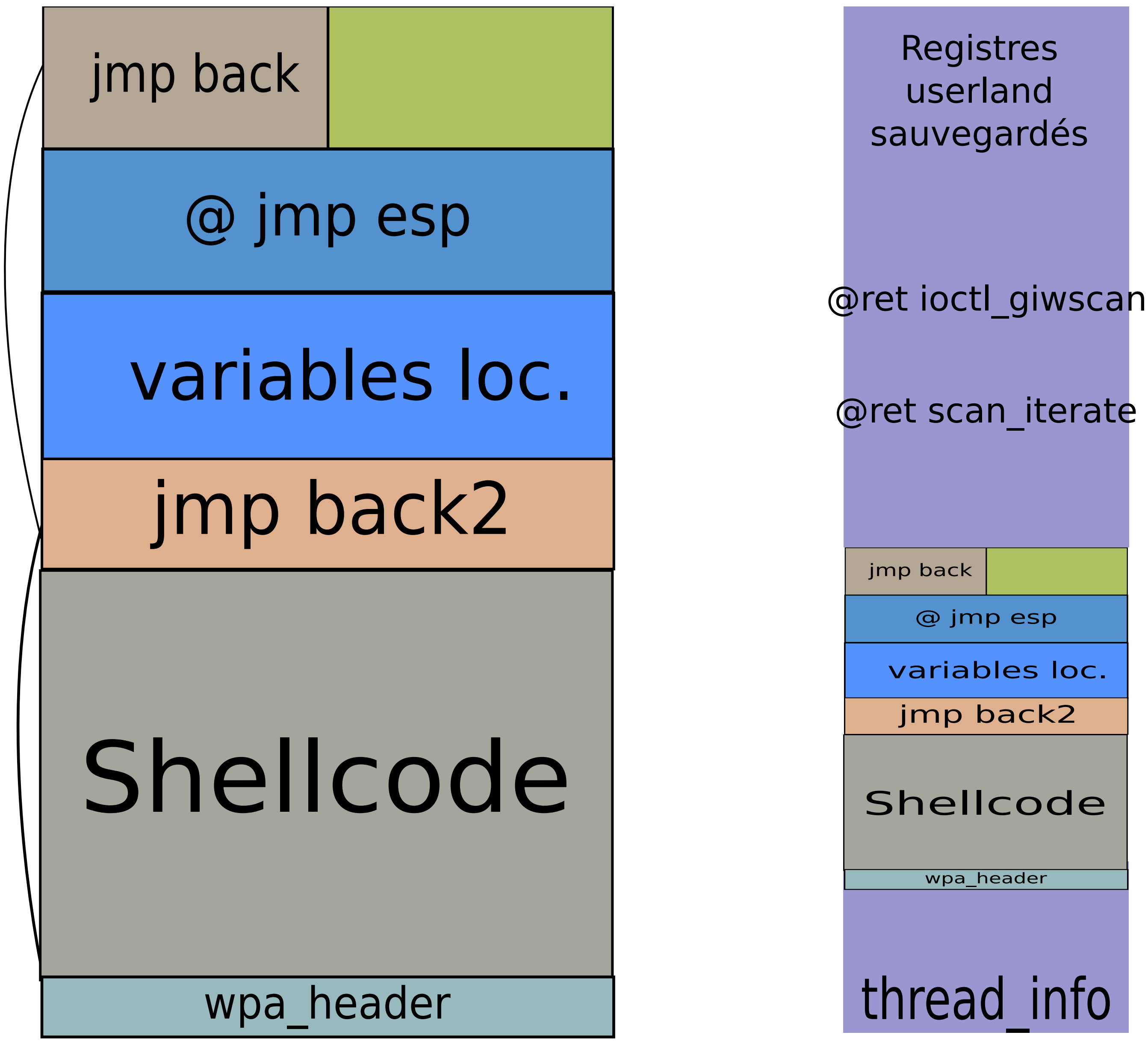
Exploitation Strategy

- Code injection in the address space
  - Let's use the 802.11 frame
  - Our information element in on the kernel stack of the current process
  - This is where we will put our shellcode

Exploitation Strategy

- Control of the execution flow
  - It seems natural to overwrite the saved EIP on the stack
  - With the address of some jmp esp
  - We can look for it either in user or kernel space
  - On the Linux 2.6 kernel, it's easy to find one:
    - dd if=/proc/self/mem bs=4096 skip=$((0xFFFFE)) count=1 of=vdso.so
    - Between the end of the elf and the end of the page, we find a jmp esp
    - It does't depend on the kernel or process version

Kernel Stack



The problem of the Kernel-mode shellcode

- Let's try to get back to a known situation
- Let's get back to userland
  - On top of the kernel stack, we can find the userland stack pointer
  - We copy a userland shellcode there
  - We change the value of userland's EIP
- We can now do an iret to return from the syscall
  - This gives an exploit which does'nt depend on the kernel version
  - But that kills the 802.11 stack, unfortunately

Save the Wifi

- We try to let the kernel resume his execution "normally"
  - We return to the caller of our caller
  - We emulate the epilogue of our caller
    - We restore the registers
    - We have to unlock a spinlock (ouch!)
- Our "userland" shellcode will execute when the system call returns
- The 802.11 stack is fine
- We could even let the process resume normally in userland!

Result

- We wrote a module for Metasploit (using Metasm) exploiting any Linux machine with an Atheros card scanning for networks
- Two different kinds of targets:
  - A very generic target, that works everywhere, but kills the 802.11 stack
  - Some more specific targets that will cleanly restore the 802.11 stack
    - An example would be Ubuntu 6.10
  - It is perfectly possible to write a multi-target exploit, since we get arbitrary code execution generically
- Our exploit can use any Metasploit payload

Demo

Did it work ?

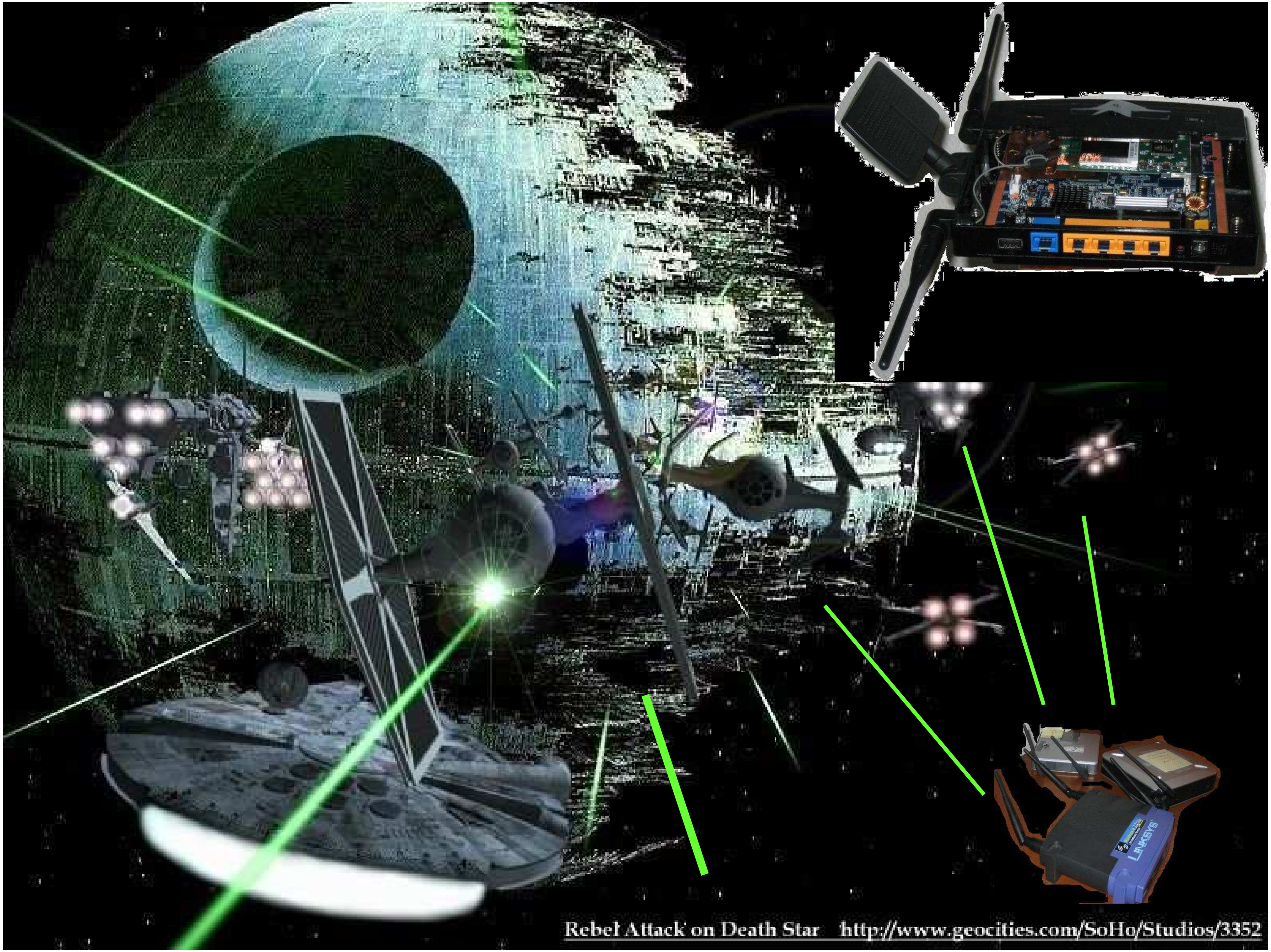
- This was the first remote kernel exploit for Linux
  - A very reliable exploit
  - Use PaX!
    - KERNEXEC against remotes
    - UDEREF against some of the local exploits
- We aim to integrate kernel payloads into Metasploit
  - For both process and interrupt context

It did’nt work ?

- Maybe someone did a DoS
- Maybe someone launched another exploit
- This is because it’s impossible to protect against this kind of flaw, even with WPA!

Fuzzing 802.11 Access Points

Access Point Wars…


LINKSYS
Rebel Attack on Death Star http://www.geocities.com/SoHo/Studios/3352

Access Points Vulnerabilities?

- Access points are embedded devices relying on wireless chipsets
- Remember wireless client implementation flaws…
- Is it possible to discover implementation bugs in access points?
- This part will describe 802.11 implementation flaws in access points
  - Thus only from the wireless side of course!

So What? A.P. Vulnerabilities?

- Attacks from any unauthenticated malicious users
  - From the wireless side even with WPA/WPA2 (with PSK or EAP)
  - Another risk for wireless enabled architectures (enterprises…)

- At least denial of service

- Possibly, remote code execution
  - MIPS, ARM architectures
  - Debugging is harder

Fuzzing 802.11 Access Points

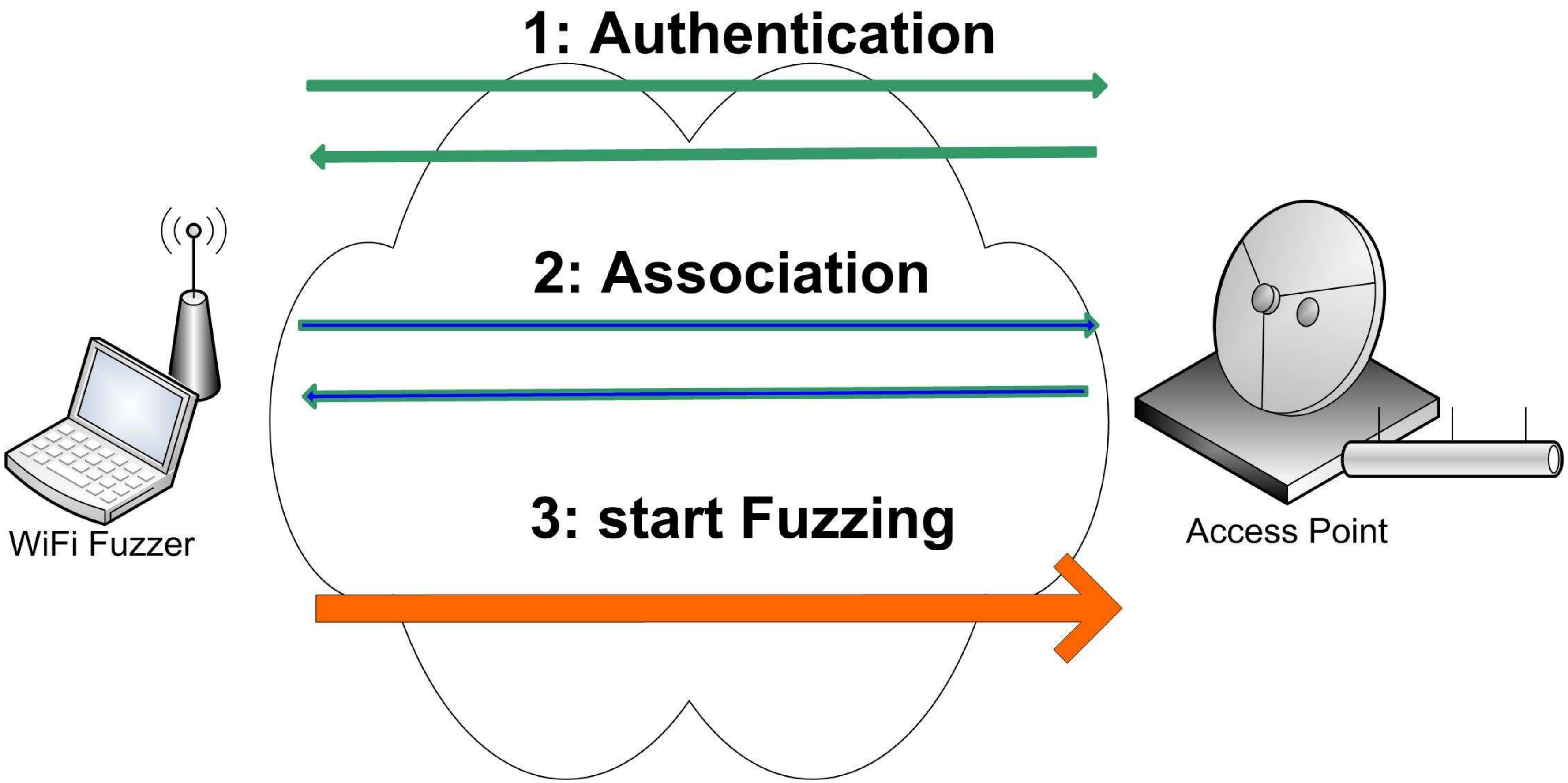
- Similar to 802.11 client fuzzing

- But better be stateful to be effective
  - Wireless client capabilities are parsed by the access point
    - During association (association requests)
    - Not during active scanning (probe requests)

Fuzzing 802.11 Access Points

- 802.11 access point stacks will parse lots of 802.11 packets
  - Probe requests
  - Authentication requests
  - Association requests
  - Crypted and unencrypted data frames
  - Control frames
- Other protocols are used in access points thus could be fuzzed
  - WPA/WPA2 key exchanges (handshakes)
  - EAP-based authentication
- Stateful fuzzing
  - Pass state 1 thanks to a successful authentication request
  - Pass state 2 thanks to a successful association request
  - If WPA/WPA2
    - Pass over state 3 thanks to a successful EAPoL-Key exchange

Stateful Fuzzing



Apwifuzz

- Based on Phil's Scapy (Python)
  - For frame forging and injection
- Generates a set of tests for any state to be fuzzed
  - Information elements fuzzing, truncated frames…
- Checks the access point configuration
  - Open, WEP, WPA/WPA2, PSK/EAP

Apwifuzz

- Launches all tests sequentially for any states
  - Perform state changes verification (successful authentication…)
- Checks if the access point is still alive after a particular test
  - Perform an “Open” authentication
- If not responsive, stop and wait for the AP to resume
  - Printing the test that triggered the bug
- Etc…

Fuzzing Access Points

- Consequences?
  - Reboot
  - Freeze: requires a manual reboot (watchdog?)
  - Reboot with wireless interface inactive
    - No more attacks ☺
- This complexifies the fuzzing process
  - Fuzzer will stop on the first bug found
  - Quite annoying

Current Status of 802.11 Access Points Vulnerabilities

- Today's access points vulnerabilities are… quite classic
  - Flaws in embedded services (httpd, cgi scripts…)

- AP 802.11 related vulnerabilities found at http://cve.mitre.org
  - CVE-2007-5448: madwifi xrates element overflow
  - CVE-2007-2829: madwifi-based vulnerability on the parsing of data frames
  - CVE-2006-2213: Malformed EAPoL-Key causes hostapd 0.3.7-2 to crash

Discovered Vulnerabilities (Access Point Implementations)

Discovered Vulnerabilties

- Cisco access points (to be detailed)
- Check reserved CVE-2007-5474 and CVE-2007-5475
- And a lot of ongoing investigations

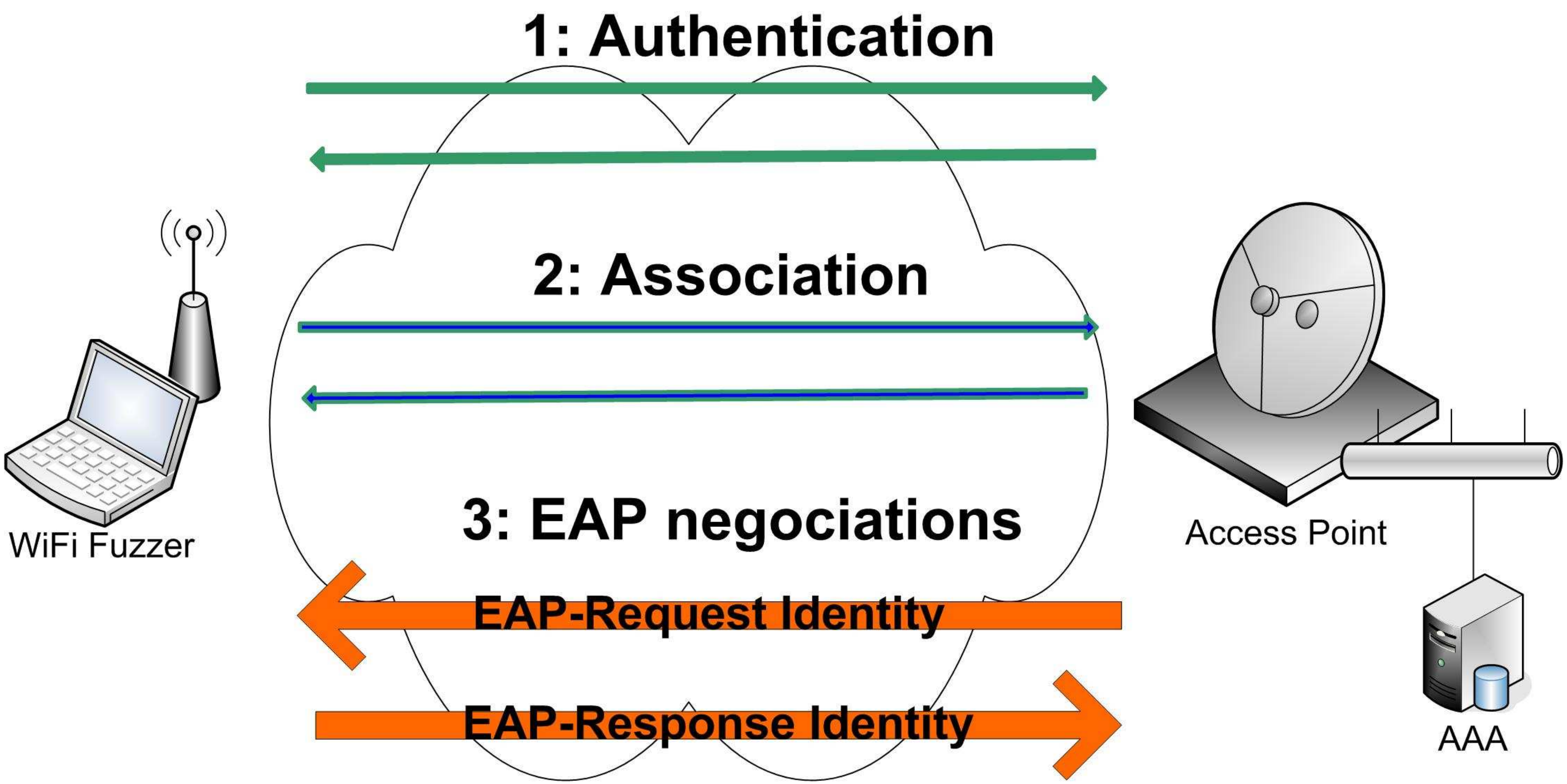
Example of an Access Point Vulnerability

- Timeline
  - 1. Authentication
  - 2. Association
  - 3. Any EAP-based packet with a short advertised length will cause the access point to crash/reboot

- The implementation incorrectly assumes that any EAP packet has a minimal length of 5 bytes
  - This field may be manipulated by a wireless attacker

- Triggered by a malformed EAP-Response Identity
  - Needs WPA/WPA2 with EAP authentication enabled

Stateful Fuzzing



 research & development

Example: the Cisco AP Vulnerability

- Impacts
  - Denial of service on any vulnerable wireless access point
  - Possible remote code execution
  - From any *unauthenticated* malicious user
- A good example of “Security vs. Complexity”
  - Even robust security mechanisms may induce issues on security
    - Implementation bugs!
- Discovered during EAP-based fuzzing of
  - A wireless access point and an EAP-based RADIUS server (EAP-TLS)

Timeline

- Vendor notified: July, 1st 2007
- Vendor acknowledged the notification: July, 1st 2007
- Details of the vulnerability explained with exploit code (private release for Cisco): July, 2nd 2007
- Cooperative work on the corrective patch: July, 2007
- Agreement on the disclosure of the vulnerability: September, 10th 2007
- Disclosure: October, 19th 2007 – **TODAY** ☺

- We thank the Cisco PSIRT team for their responsiveness

Side Effects and Disclosure

- It impacts lots of devices
  - Not only wireless access points
  - This is an EAP-based vulnerability
  - Wired switches with 802.1X/EAP enabled may be vulnerable
- Also other vendors/products may be vulnerable as it is a generic vulnerability
- Cisco's official advisory is planned to be published in classic mailing lists today
- … check your mails … and patch!

Investigations on Exploitability

Investigating APs Vulnerabilities

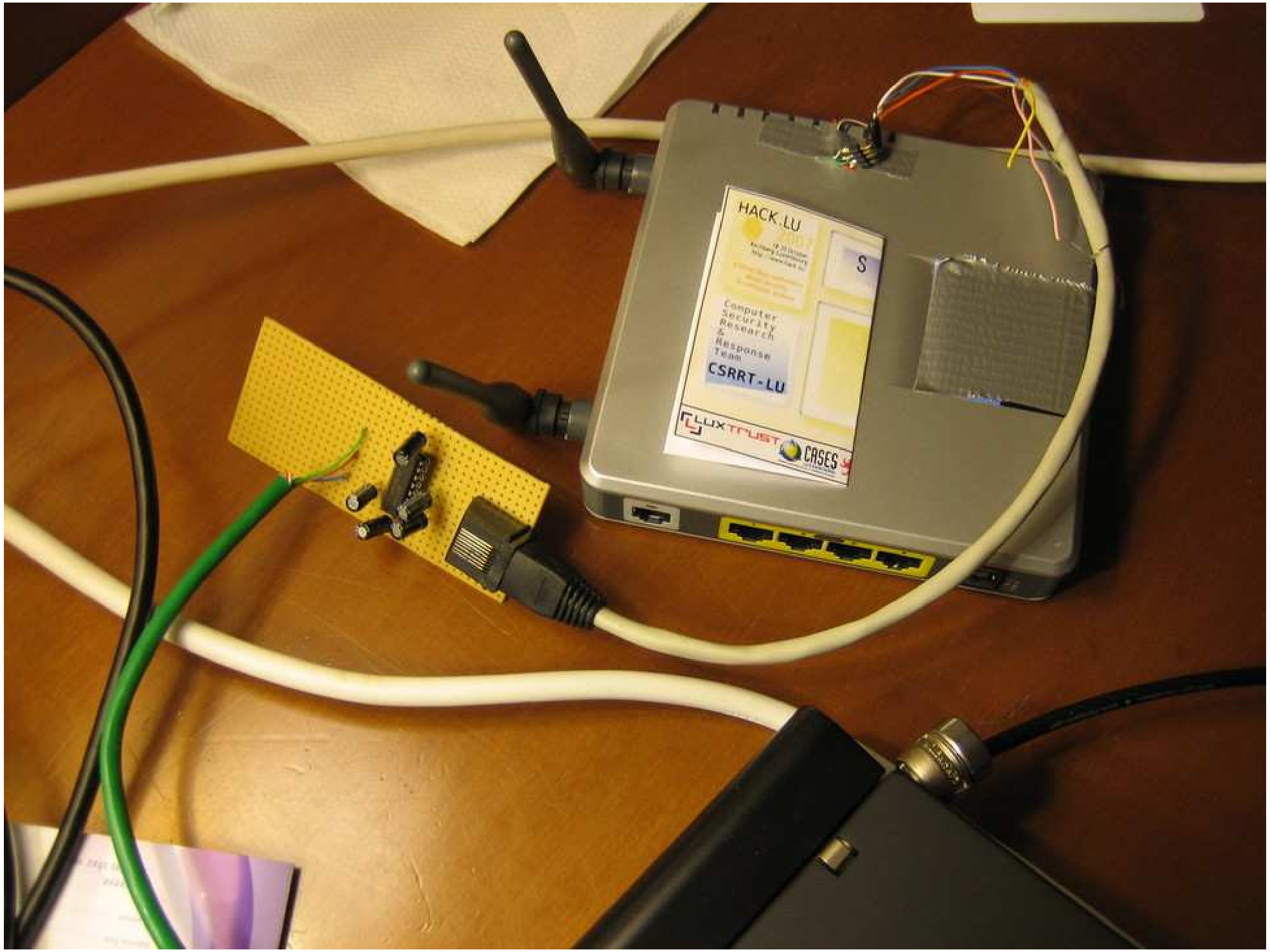
- We can remotely crash a lot of access points
  - We have a fairly good success rate
- We need more information
  - Nature and localisation of the flaws
  - Exploitability to gain remote control over the access point
- We can write a small DoS exploit to easily trigger the vulnerability
  - Based on the information given by the fuzzer

Getting some information

- Open the box, look at the board
  - Look at SoC on the main board and determine the architecture
  - Look at the Wifi chip (Atheros, Marvell…)
  - Google

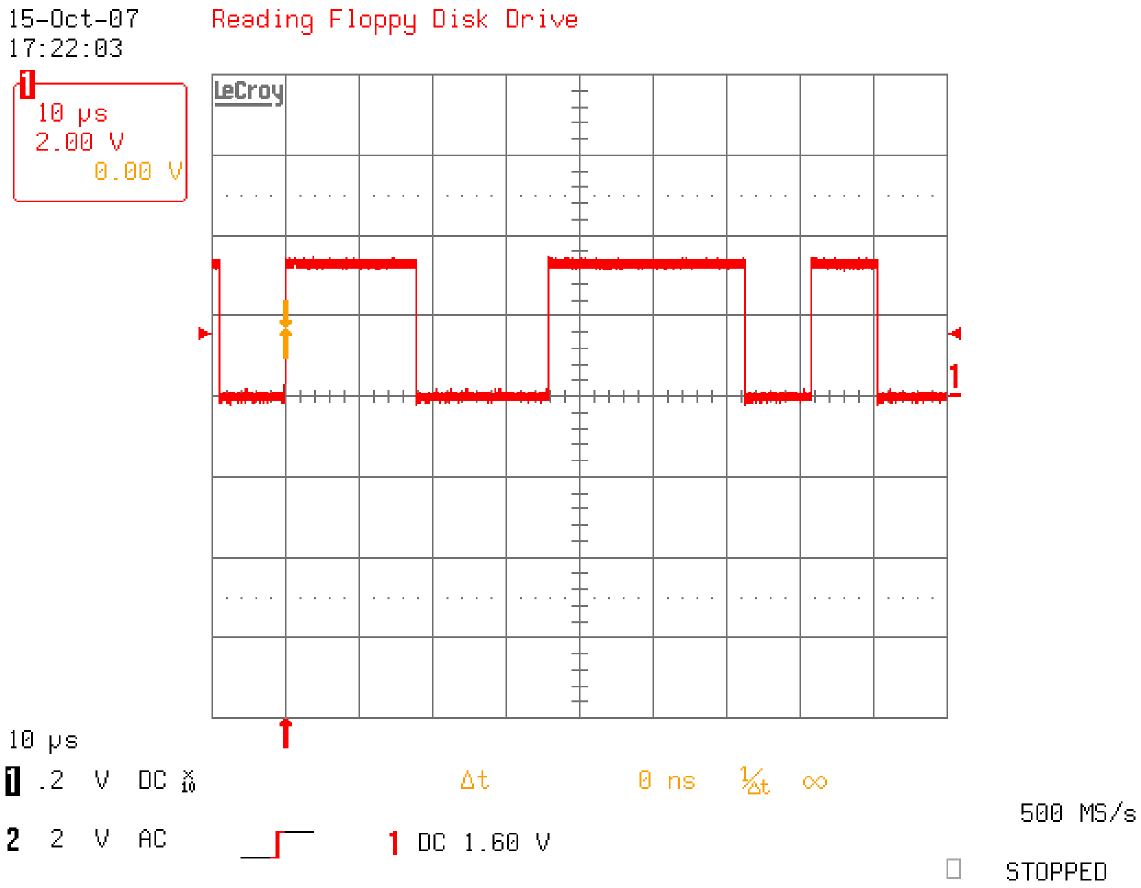
- Some access points will let you get a shell easily
  - Standard, externally accessible serial port
  - Telnet server
- Sometimes it can be a trickier
  - Internal serial port (need for TTL->RS232 conversion)

TTL->RS232


HACK.LU
Computer Security Research & Response Team
CSRRT-LU
CASES

Finding the serial ports

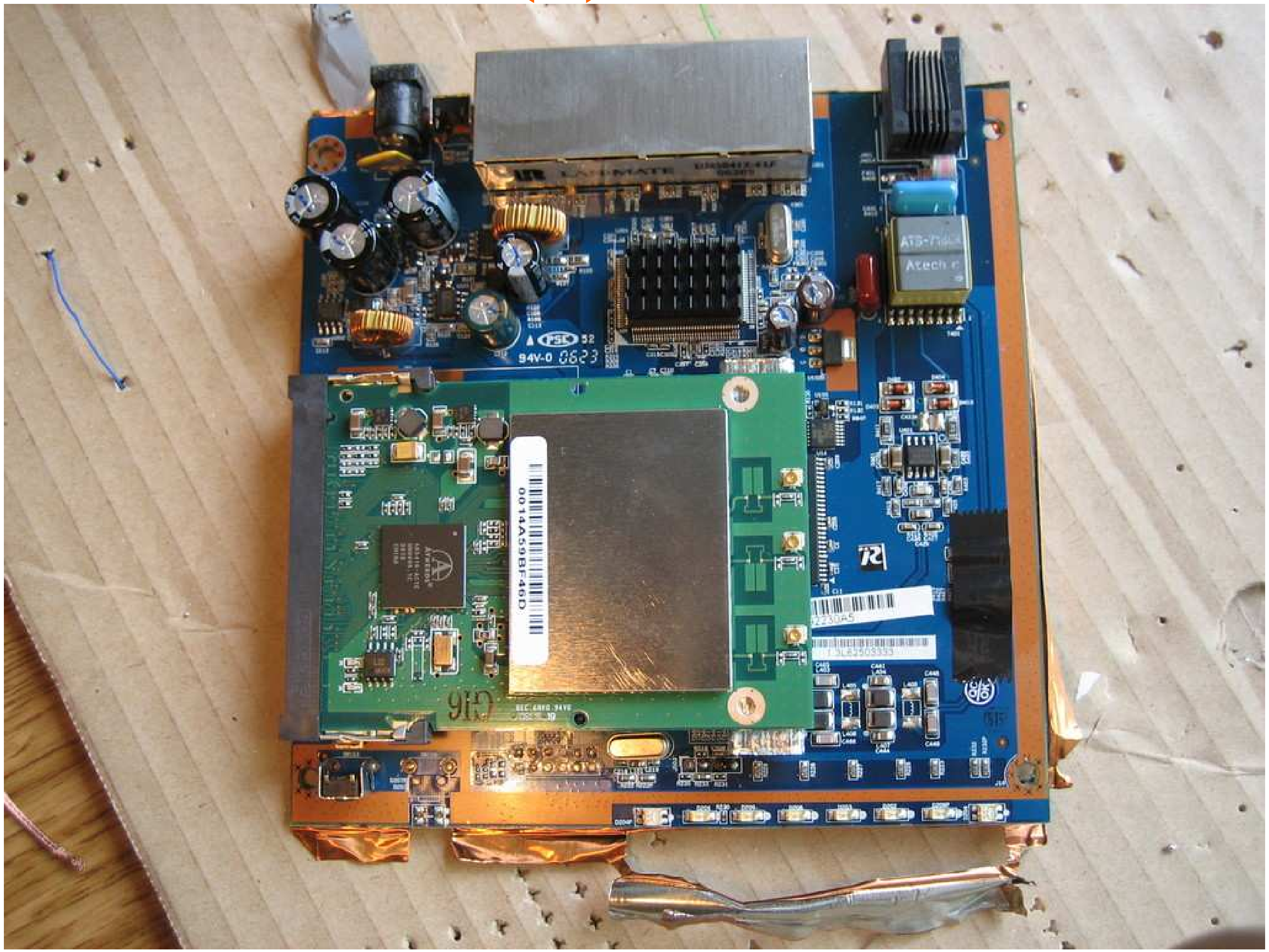
- Using a multimeter
  - Find VCC and Ground
- Using an oscilloscope
  - Find TX



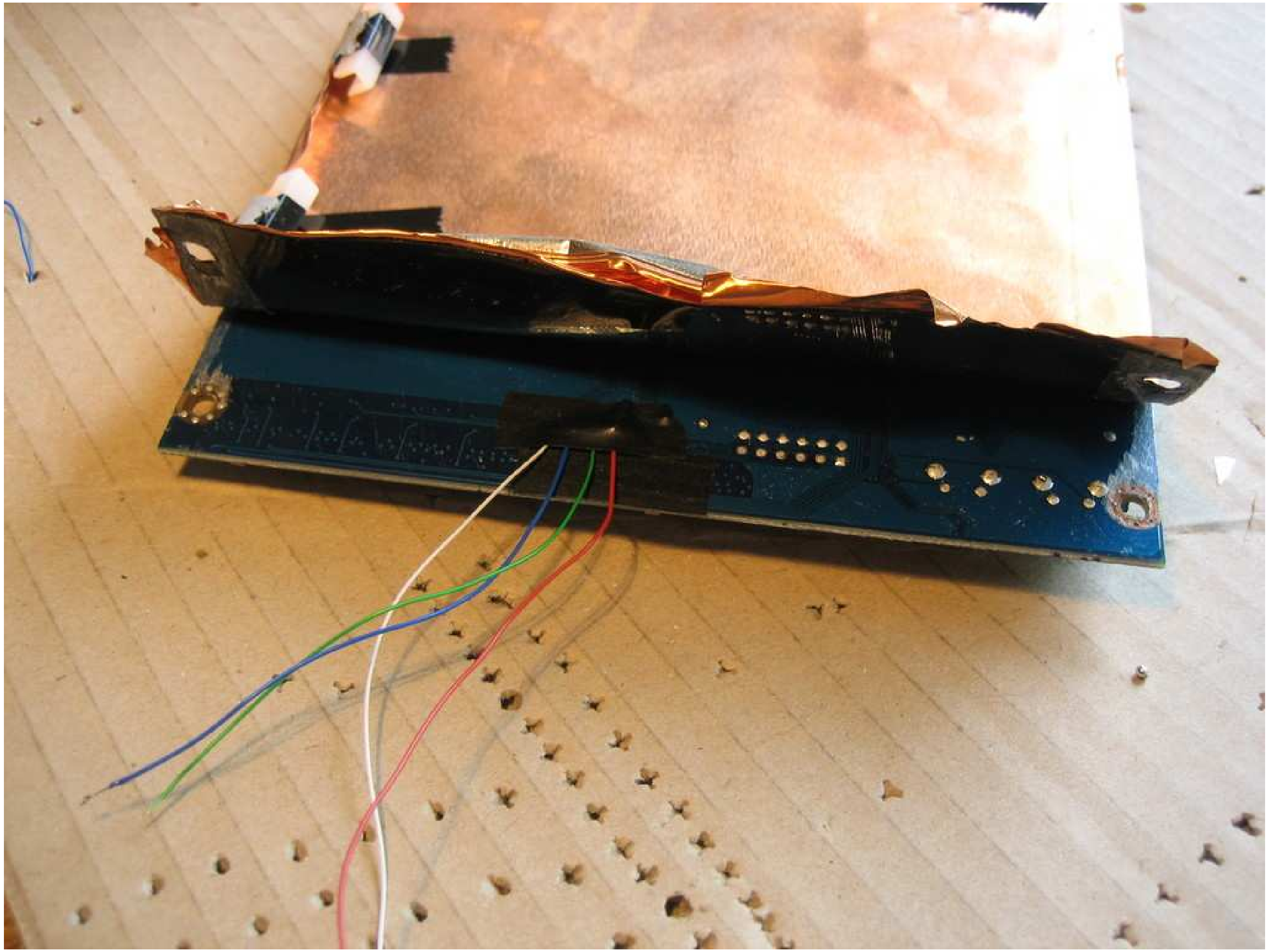
Easy version

Harder version

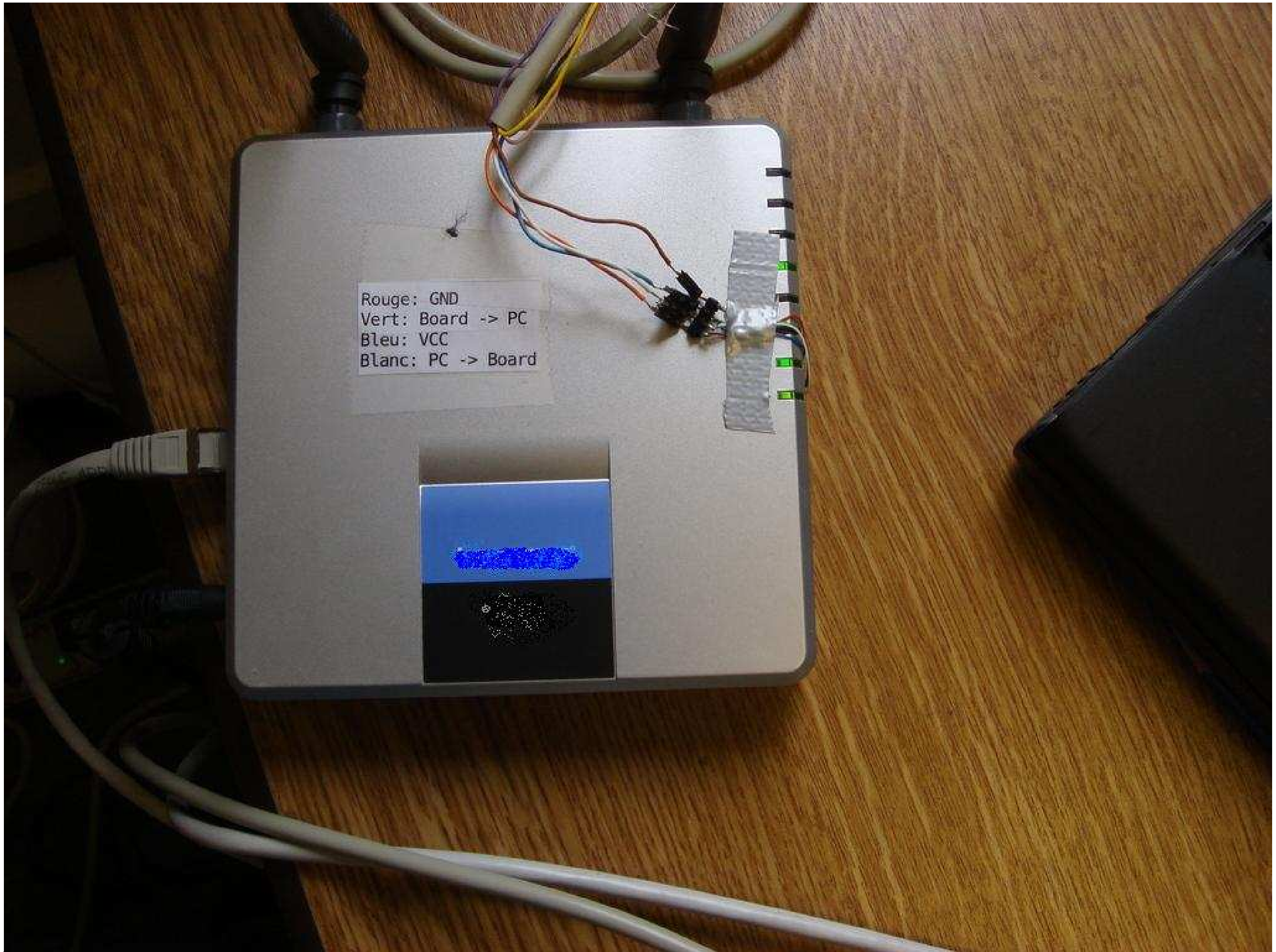
Harder version (2)



Harder version (3)



Harder version (4)



Getting a shell

- If you can’t find a serial port or if the serial console prompts for a password
- Get a firmware update file
  - Usually easy to decipher (mostly simple, non cryptographic algorithms, easy to guess)
  - Unpack it (squashfs or cramfs for Linux-based devices)
- If that doesn’t work, use JTAG to dump the flash memory
- Once you get the firmware
  - Find exploitable bugs (Web server, configuration restoration process)
  - Find hidden debug features
  - Modify it if you can
- Last resort
  - Use the JTAG to patch firmware in flash

Hidden debug feature in an AP

IDA View-A | Hex View-A | Exports | Imports | Names | Functions | Strings | Structures | Enums

```
.text:00409C08     sw      $ra, 0x20+var_8($sp)
.text:00409C0C     sw      $gp, 0x20+var_10($sp)
.text:00409C10     lw      $a0, offset 0x435ac8
.text:00409C14     lw      $t9, offset system
.text:00409C18     nop
.text:00409C1C     jalr    $t9
.text:00409C20     addiu   $a0, 0x5AC8      # "/usr/bin/killall utelnetd"
.text:00409C24     lw      $gp, 0x20+var_10($sp)
.text:00409C28     nop
.text:00409C2C     lw      $a0, offset 0x435ae4
.text:00409C30     lw      $t9, offset COMMAND
.text:00409C34     nop
.text:00409C38     jalr    $t9
.text:00409C3C     addiu   $a0, 0x5AE4      # "/usr/sbin/utelnetd -d &"
.text:00409C40     lw      $gp, 0x20+var_10($sp)
.text:00409C44     nop
.text:00409C48     lw      $a0, offset aTextHtml
.text:00409C4C     lw      $t9, offset mime_header
.text:00409C50     nop
.text:00409C54     jalr    $t9
.text:00409C58     addiu   $a0, 0x4114      # "text/html"
.text:00409C5C     lw      $gp, 0x20+var_10($sp)
.text:00409C60     nop
```

Debugging the flaw

- Try to get some kind of backtrace or information
  - OOPS() on Linux
  - Sometimes you'll even get symbols
  - Demo

Conclusions

Conclusions

- 802.11 extensions are complex thus error-prone
- Even if not exhaustive, 802.11 fuzzing is an effective technique
- We found some critical bugs
  - Stay tuned for more…
  - WPA/WPA2 will not protect your client or infrastructure
- Successful access point exploitation may be available soon!

Acknowledgements

- Yoann Guillot for metasm
- Raphael Rigo for help on access point investigations
- Benoit Stopin for the development of the EAP fuzzer and the discovery of the Cisco access point bug

May The Force Be With You