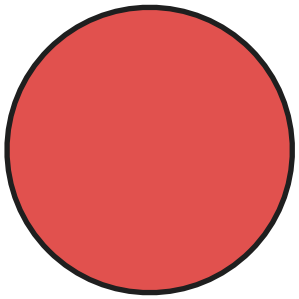


Stateful Fuzzing of Wireless Device Drivers in an Emulated Environment

Tokyo
25 October 2007

sk [at] seclab [dot] tuwien [dot] ac [dot] at — Sylvester Keil

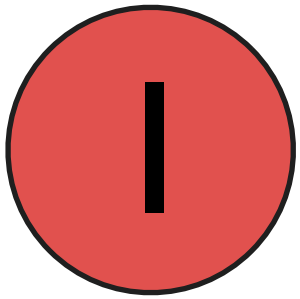
ck [at] seclab [dot] tuwien [dot] ac [dot] at — Clemens Kolbitsch

About us

- We are two students from the **Technical University Vienna** in Austria
- Right now we ought to be working on our master theses at the **Secure Systems Lab @ TU Vienna**
- The work presented here is a collaboration between the **Secure Systems Lab** and **SEC Consult**

http://www.seclab.tuwien.ac.at

http://www.sec-consult.com
research [at] sec-consult [dot] com



Introduction

Introduction

- Wireless networks have become a widely spread means of communication. Compatible devices are included in most portable computers, printers, mobile phones …
- Publicly available networks and "hot spots" are becoming increasingly popular
- Typically, wireless devices are turned on even if they are not used
- Network drivers scan for available networks and continuously try to communicate with other stations

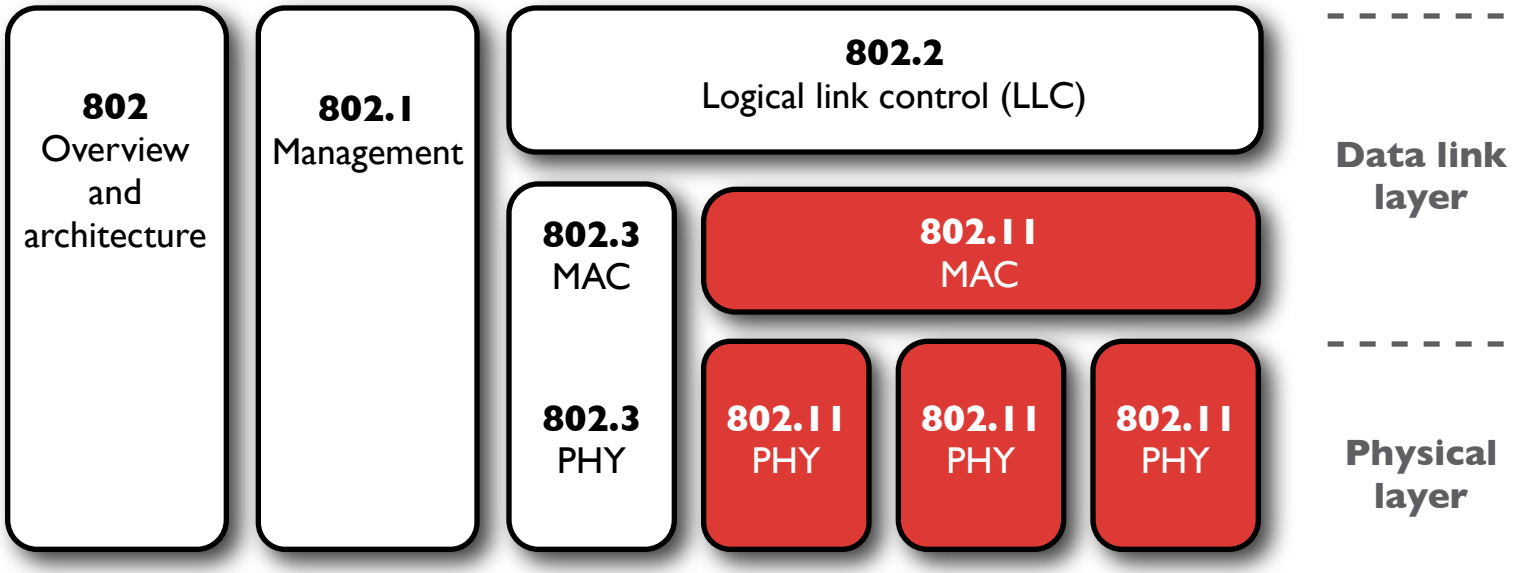
That means …

- There is an increasing number of potential mobile targets out there
- No physical access (keyboard, cable, etc.) is necessary to interact with a possible target
- What's more, the communicating software (i.e. device drivers) operates on system/kernel level, thus rendering potential vulnerabilities extremely dangerous
- The conclusion?

  Wireless device drivers can be a lot of fun!

IEEE 802.11 Fundamentals

IEEE 802 Family



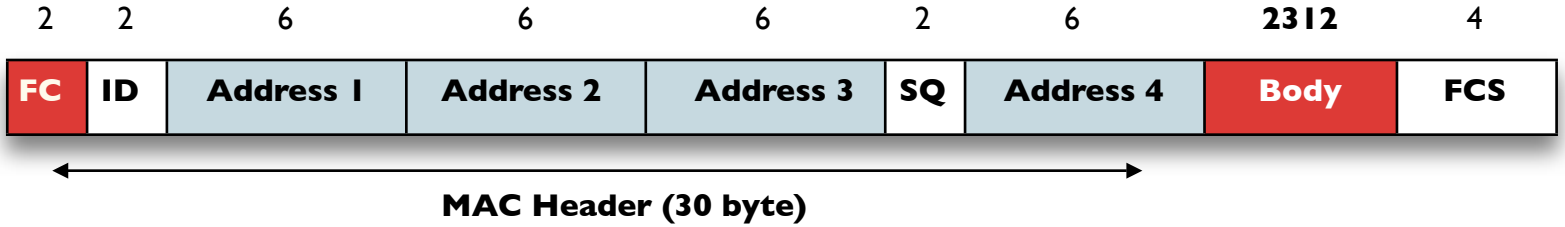
Why is 802.11 so complex?

- The standard designers faced many challenges because of the underlying physical medium:
  - **Co-ordination of participants**
  - **Distribution**
  - **Integration with wired networks**
  - **Confidentiality**
  - **Power management**

Implications

- Three different states:
  - (Not) **authenticated** and (not) **associated**
- Three different frame types:
  - **Control**, **Data** and **Management** frames
- Three different modes:
  - **Master**, **Managed**, **Ad-hoc** (and **Monitor**)
- Different frequency **channels** and **BSSIDs**

IEEE 802.11 MAC

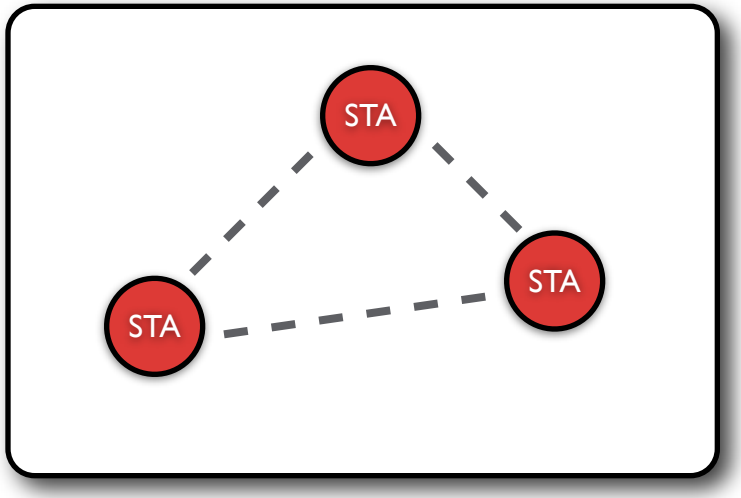


MAC Frame Control (16 bit)

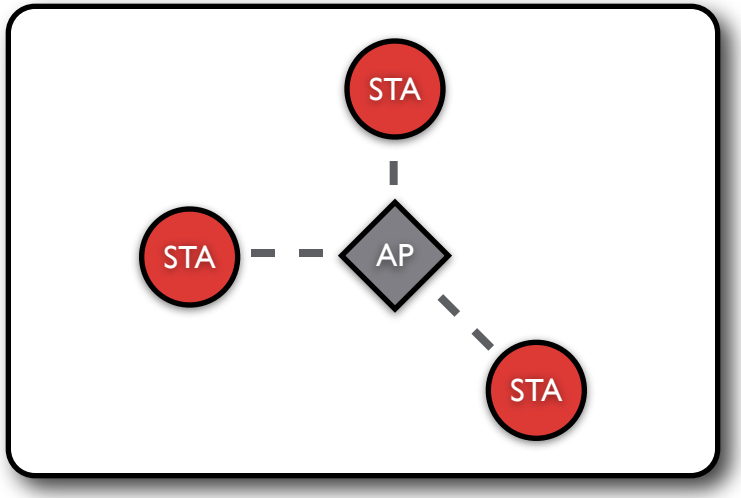
Protocol	Type	Subtype	To DS	From DS	More Frag	Retry	Pwr Mgmt	More Data	WEP	Order

Type 00 ... Management Frames
Type 01 ... Control Frames
Type 10 ... Data Frames

IEEE 802.11 Networks



Independent BSS
(**ad-hoc**)

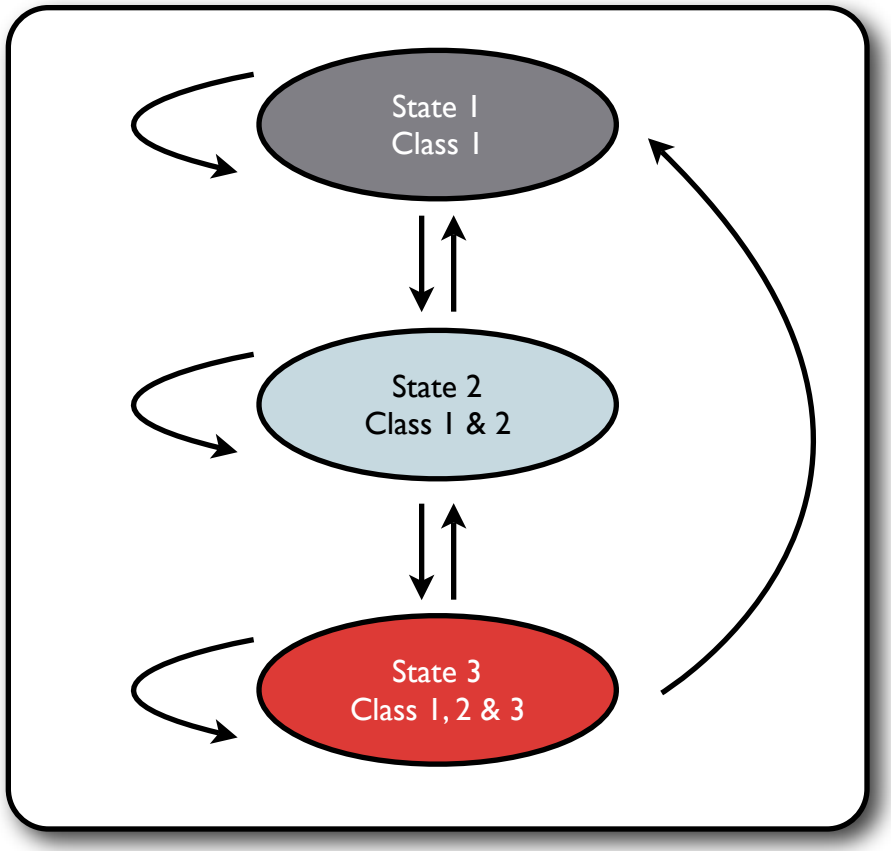


Infrastructure BSS
(**managed / master**)

IEEE 802.11 States

Successful
Authentication

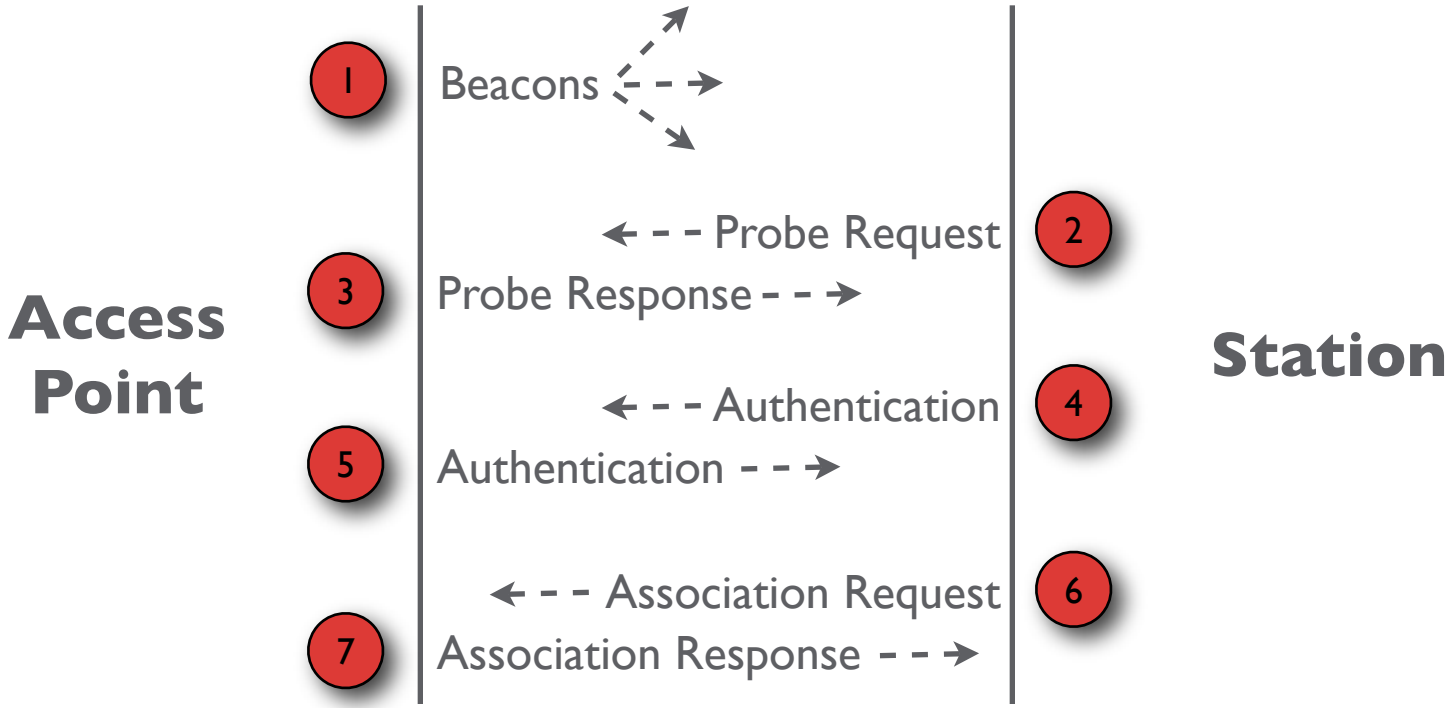
Successful
Association
or
Reassociation



DeAuthentication
Notification

Disassociation
Notification

IEEE 802.11 Association



Wireless Fuzzing

The meaning of fuzzing?

- Has become a buzz word, but originally fuzz testing or fuzzing is **software testing technique** that provides **random data** ("fuzz") as input to a program (thanks Wikipedia)
- Coined and developed at University of Wisconsin Madison in 1989
- Complexity of fuzzers may range from very simple to highly sophisticated
- Shown to be very **effective for testing of protocol implementations**

MAC Frame Injection

- Some chipsets/drivers allow transmission of raw data
- **LORCON** hides differences of underlying hardware (some drivers must be patched)
- **Scapy** supports 802.11 MAC Frame injection from Python (if the driver allows it)
- The **Metasploit Framework** 3.0 contains a Ruby interface to LORCON

Fuzzing Issues

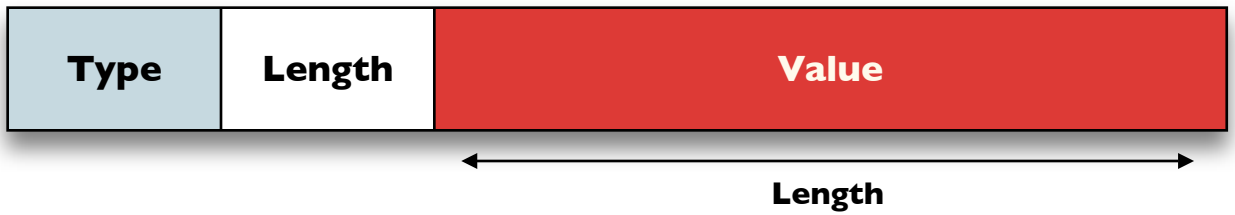
- Must be aware of different **frequencies** (**channels**), **BSSIDs**, **states**, **modes** and data link **encryption** (filtering may take place at the hardware level!)
- **Response time** and **timing** of replies is critical (i.e. short request-reply sequences)
- **Attacker and target must be co-ordinated** (mode, state etc.) and target must be **continuously monitored**
- **Overload, interference, packet corruption**

Wireless Fuzzing

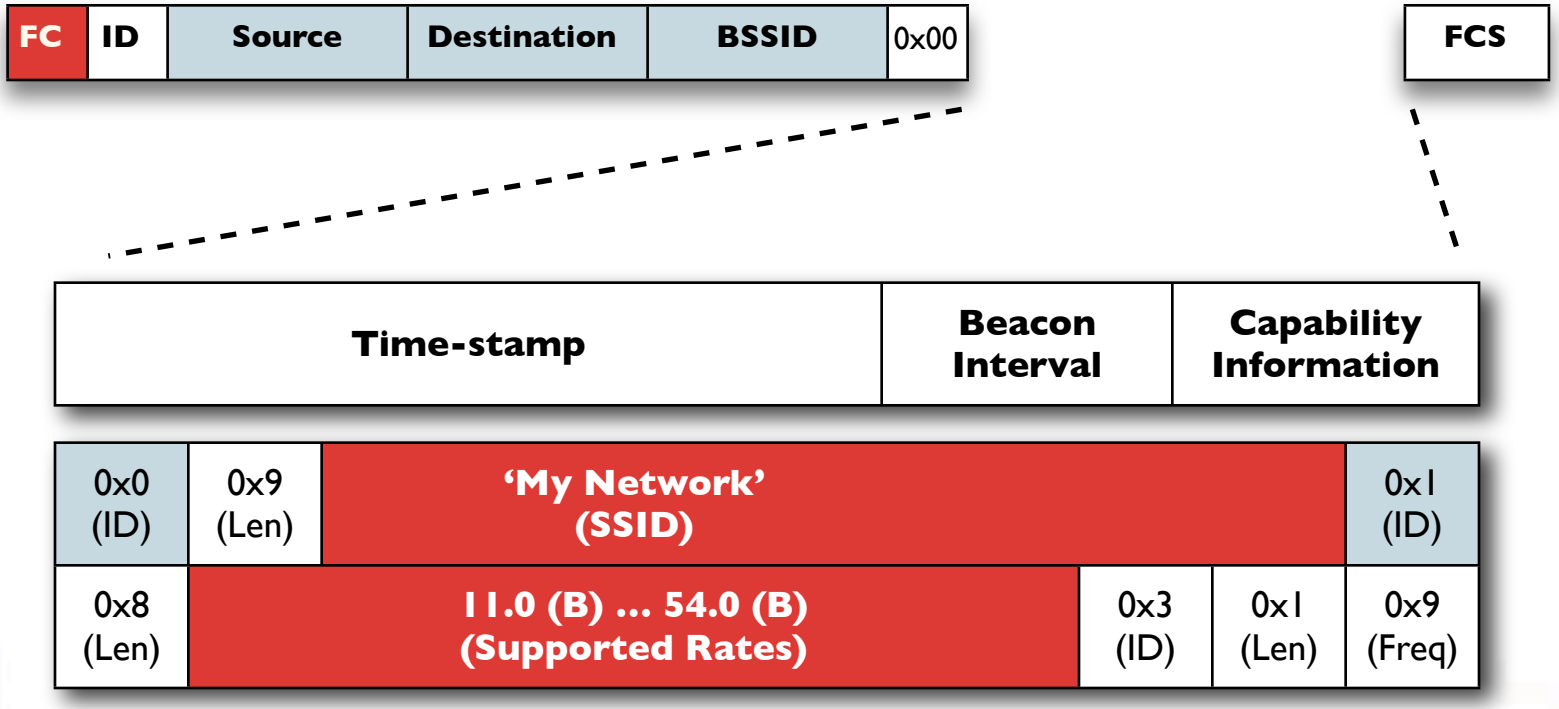
- Network identification & infrastructure
- **Information Elements (IE)** follow **type-length-value** pattern
- **Stateless fuzzing**: BSSIDs, supported data rates …
- **Stateful fuzzing**: Authentication, association, encryption and many more!
- Some subtypes are **mode-dependent** (e.g. probe requests may be accepted by targets in ad-hoc networks)

type-length-value

- This pattern is often used data communication protocols for optional information
- **Type** and **length** fields have fixed size
- **Value** field is of variable size



Example: A Beacon Frame



What to fuzz?

States	Frame type	Potentially interesting fields
1, 2, 3	Beacon	SSID, TIM, Country Info, Extended Rates
1	Probe Request	(Ad-hoc only)
1, 2, 3	Probe Response	SSID, Supported Rates, Country Info, FH Pattern, Extended Supporetd Rates, RSN
1	Authentication	
2	Association Request	(Ad-hoc only)
2	Association Response	Supported Rates
3	Re-association, Disassociation	
3	Data	
2, 3	Encryption	

Wireless Fuzzers

- A number of 802.11 fuzzers have been developed quite successfully: vulnerabilities have been found in various drivers on different platforms (e.g. the vulnerabilities presented by Laurent Butti at Black Hat Europe 2007)
- **BUT** these vulnerabilities were detected exclusively using stateless or relatively simple fuzzers
- It would be desirable to have a fuzzing framework to easily test 802.11 drivers in all states and without the hindering implications of the wireless medium!

Virtual Wireless Fuzzing

A novel approach

- **Requirements**
  - Eliminate **timing contraints**
  - Replace unstable wireless medium
  - Allow **guaranteed delivery**
  - Support advanced **target monitoring**
- **Solution**
  - **Move target into a virtual environment!**

Advantages

- **Virtual wireless device** (software) replaces network hardware
- **High-level IPC** instead of packet-injection
- **CPU** in virtual machine **can be interrupted and stopped** at all times if necessary
- Guest OS **monitoring** at **low-level** (system restart, console output, etc.)
- Drastically eliminates the complexity of "traditional" 802.11 fuzzers

Our solution

- Develop a fuzzing "framework" on the basis of Fabrice Bellard's QEMU
- You can find the framework on the Conference CD, published under the GPL
- We will also give a short presentation after the following survey

System Overview

The Atheros AR5212 Chip

- Widely used 802.11a/g chip
- Various Windows drivers
- MadWifi Linux driver (with binary HAL)
- OpenHAL project

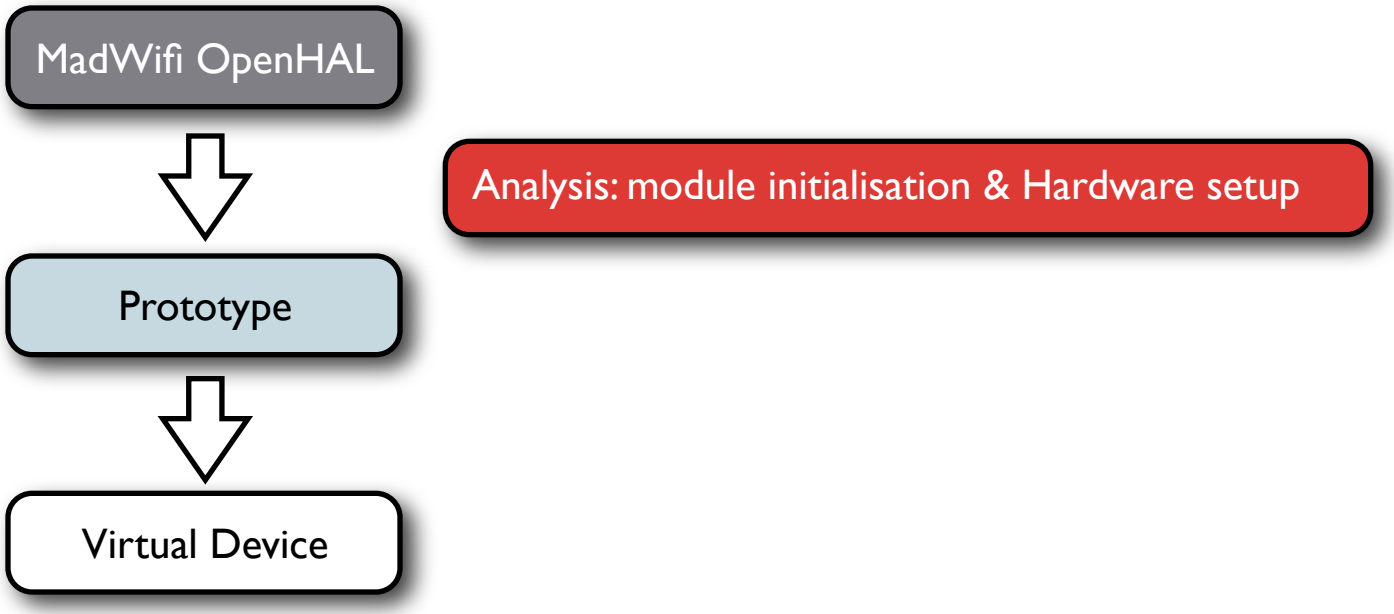
AR5212 Details

- Very powerful (may use prohibited frequencies)
- Communication through **memory mappings** and **interrupts**
- **four outgoing queues** (only one used in virtual device)
- **one incoming queue**
- EEPROM (accessed through magic numbers on device memory)

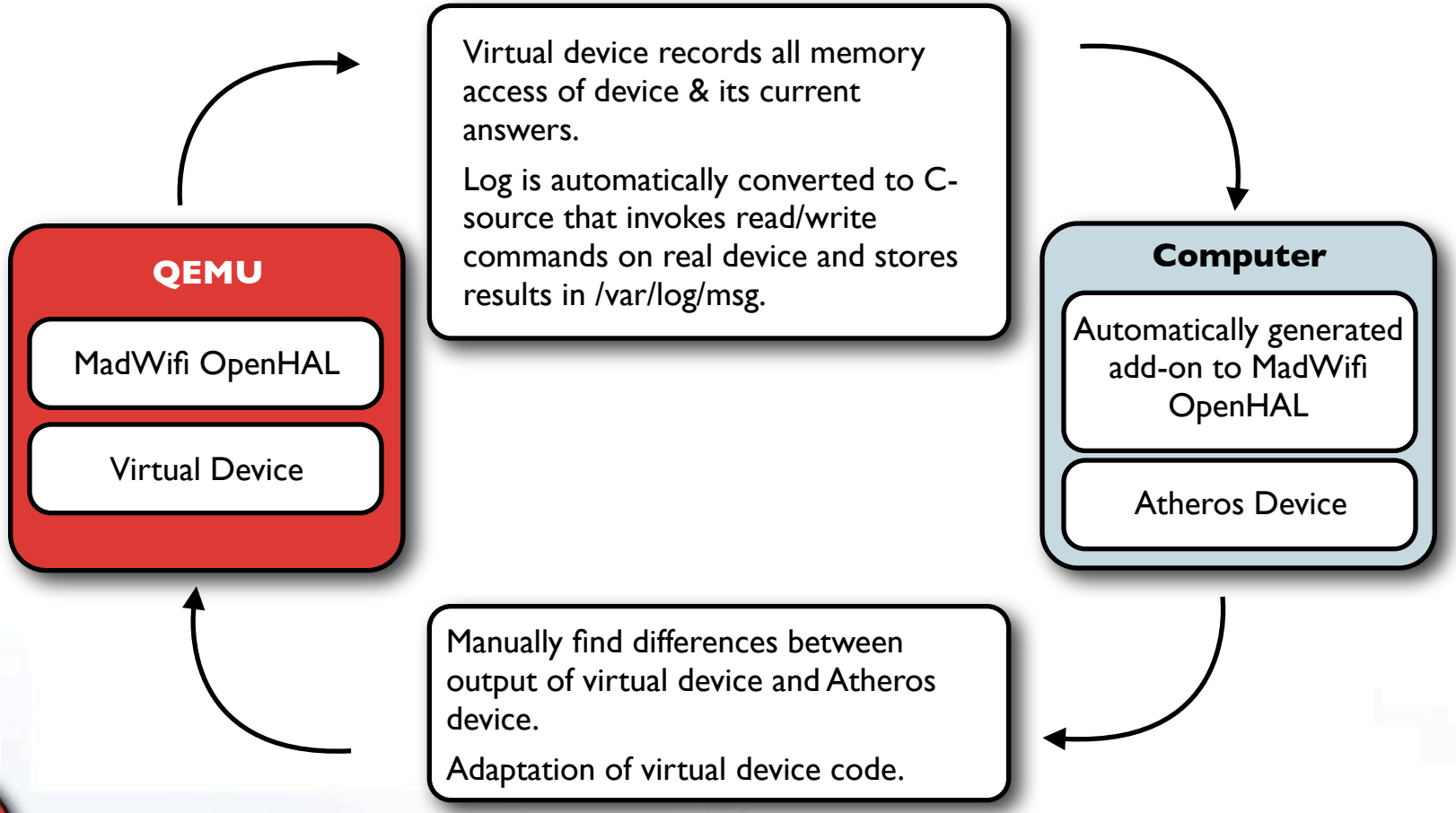
Reverse Engineering

- Comprises 4 sub-tasks
  - Hardware initialisation & chip status test (specifically with Windows driver)
  - Detection of memory writes to outgoing queue & reading memory mapped data regions
  - Interrupt generation (interrupt masks)
  - Injection of data into incoming queue / filling memory mapped data regions
- Initial process based on MadWifi OpenHAL
- Iterative process based on real Atheros device

Reverse Engineering



Reverse Engineering



Reverse Engineering

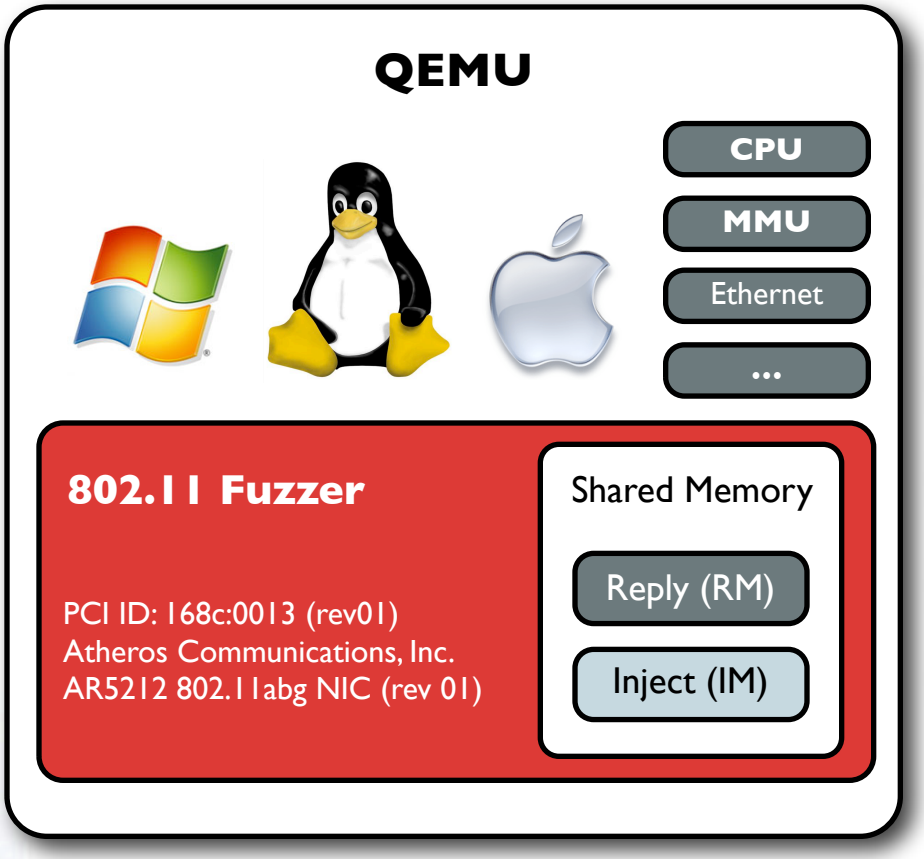
- Focus on necessary functions (send/receive, set channel)
- Some functions partly reverse engineered, but not used (e.g. set a/b/g mode)
- Some additional functions totally ignored

- Nice by-product: differences between binary and open HAL can be logged

The Virtual Device

- Optional hardware/ethernet card, can be added through QEMU command line option
- Windows/NDIS-wrapper and MadWifi version
- Modular design
  - packets read from outgoing queue are written into shared memory
  - connected modules are notified via semaphores
  - packets are read from shared memory and inserted into incoming queue

System Design



- Dumper [RM]: store outgoing packets
- Listener [RM]: display outgoing packets
- Injector [IM]: inject arbitrary packets
- Stateless Fuzzer [IM]: reply directly
- Access Point [RM] & [IM]
- Stateful Fuzzer [RM] & [IM]: AP and Fuzzer

Module: Access Point

- Broadcasts **Beacon** frames
- Responds to incoming **Probe Requests**
- Complete Basic Authentication
- Responds to incoming **Association Requests**
- Features minimum implementation of ICMP
- Full logging of 802.11 communication
- But words can only say so much …

Module: Stateful Fuzzer

- Initially the fuzzer behaves like the access point module, broadcasts valid **Beacon** frames and responds to incoming **Probe Requests**
- Once authentication is complete, it is possible to fuzz the target in the authenticated / not associated state, e.g. transmit fuzzed **Association Response** frames
- See it yourself …

Results & Conclusion

Results

- We have developed a "framework" for fuzzing 802.11 device drivers using QEMU
- So far the framework supports fuzzing of **all three states** of a target in **managed mode**
- A simple fuzzer using the framework and old versions of the MadWifi driver detected known vulnerabilities
- A **previously undocumented vulnerability** in the newest version of the driver was also found!

The Vulnerability

- Our fuzzer detected a flaw in the current MadWifi implementation
- A **Beacon** frame with a specially crafted **Extended Supported Rates** Information Element crashes Linux when scanning for available networks
- Sadly, no remote code execution possible (only DoS)
- Recently published by **SEC Consult & TU Vienna**

```
   Type/Subtype: Beacon frame (8)
 ▸ Frame Control: 0x0080 (Normal)
   Duration: 0
   Destination address: Broadcast (ff:ff:ff:ff:ff:ff)
   Source address: Cisco-Li_c7:87:c1 (00:0f:66:c7:87:c1)
   BSS Id: Cisco-Li_c7:87:c1 (00:0f:66:c7:87:c1)
   Fragment number: 0
   Sequence number: 0
   Frame check sequence: 0x4f86df1d [correct]
▾ IEEE 802.11 wireless LAN management frame
 ▸ Fixed parameters (12 bytes)
 ▾ Tagged parameters (273 bytes)
   ▸ SSID parameter set: "Vuln"
   ▸ Supported Rates: 1.0(B) 2.0(B) 5.5(B) 11.0(B) 18.0 24.0 36.0 54.0
   ▸ DS Parameter set: Current Channel: 9
   ▾ Extended Supported Rates: 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
       Tag Number: 50 (Extended Supported Rates)
       Tag length: 252
       Tag interpretation [truncated]: Supported rates: 0.0 0.0 0.0 0.0 0.0
```

```
0000  00 00 10 00 0e 18 00 00  10 02 94 09 a0 00 00 26   ........ .......&
0010  80 00 00 00 ff ff ff ff  ff ff 00 0f 66 c7 87 c1   ........ ....f...
0020  00 0f 66 c7 87 c1 00 00  8d 61 a5 18 00 00 00 00   ..f..... .a......
0030  64 00 01 00 00 04 56 75  6c 6e 01 08 82 84 8b 96   d.....Vu ln......
0040  24 30 48 6c 03 01 09 32  fc 00 00 00 00 00 00 00   $0Hl...2 ........
0050  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00   ........ ........
0060  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00   ........ ........
0070  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00   ........ ........
0080  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00   ........ ........
0090  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00   ........ ........
00a0  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00   ........ ........
00b0  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00   ........ ........
00c0  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00   ........ ........
```

Conclusion

- Fuzzing 802.11 on the air is a cumbersome and time-consuming process because of the limitations and requirements of the wireless medium
- Moving the fuzzer and its target into an emulated environment **dramatically speeds up and simplifies** the process!
- Writing a 802.11 fuzzer using our framework is **easy and fast** ;-)

References & Tools

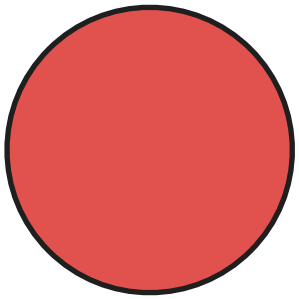
Laurent Butti. "Wi-Fi Advanced Fuzzing"
Black Hat, Europe 2007

Fabrice Bellard. "QEMU, a Fast and Portable Dynamic Translator"
USENIX 2005 Annual Technical Conference

Wireshark
http://www.wireshark.org

QEMU
http://www.qemu.org

AutoIt
http://www.autoitscript.com



Kudos & Respect

Christopher Kruegel
Engin Kirda
http://www.seclab.tuwien.ac.at

Bernhard Müller
http://www.sec-consult.com

どうもありがとう

thank you