ERNW
Living Security.

Layer 2 Fuzzing

Daniel Mende & Simon Rich
{dmende,srich}@ernw.de

Notice

- **Everything you are about to see, hear, read and experience is for educational purposes only. No warranties or guarantees implied or otherwise are in effect. Use of these tools, techniques and technologies are at your own risk.**

Agenda

- **Types and Concepts of Fuzzing**
- **Fuzzing Landscape & Options**
- **The Need for a Layer2 Fuzzer**
- **Let‘s go practical then**
  - *MPLS*
  - *VTP*
  - *DTP*
  - *EDP*
  - *WLCCP*
  - *LLDP*

Who we are

ERNW
Living Security.

- **We work as security researcher for Germany based ERNW GmbH.**
- **Fiddling around with hardware and low level protocol stuff makes the majority of our days.**
- **We were contributed to finding several protocol flaws in the past and are known for innovative approaches to implementing or breaking the security of technologies**

Definition

ERNW Living Security.

- **“Fuzz testing or Fuzzing is a Black Box software testing technique, which basically consists in finding implementation bugs using malformed/semi-malformed data injection in an automated fashion**
  http://www.owasp.org/index.php/Fuzzing

- **“A highly automated testing technique that covers numerous boundary cases using invalid data (from files, network protocols, API calls, and other targets) as application input to better ensure the absence of exploitable vulnerabilities.”** *Peter Oehlert, “Violating Assumptions with Fuzzing”, IEEE Security & Privacy, March/April 2005*

Fuzzing Landscape & Options

ERNW Living Security.

- **Quite some fuzzers/frameworks available**
- **Most of them: unmaintained or one-man projects**
- **Interesting Fuzzing Frameworks**
  - **PEACH**
  - **autodafe**
  - **scapy**
  - **proxyFuzz**
  - **GPF – General Purpose Fuzzer**
    - **With Evolutionary Fuzzing System (EFS)**
  - **SPIKE**
  - **Sulley**

The Need for a Layer 2 Fuzzer

ERNW Living Security.

- **So far nothing available in the “free tool space”.**
- **Quite some options in commercial space (think of *BreakingPoint, Mu, Codenomicon* et.al.), but all these *very* pricey.**
- **Multi purpose L2 packet crafter(s) out there (mainly *yersinia*)… but the focus of those tools is – regarding accuracy in fulfilling specifications – completely different from that of a fuzzer ;-)**

Why did we jump into this field?

ERNW Living Security.

- **See above: know the feeling “it would be nice to have a tool at hand that does...“ ?**

- **To gain some understanding of the way network fuzzers (and frameworks) work.**
- **Gain some understanding of specific protocols.**
  - => so far we mostly implemented “exotic protocols“ (e.g. no STP...)
- **To be able to “get an impression“ of a device‘s robustness in a given scenario.**

- **Not (too much): vulnerability research. We did not try to find the exact parser weaknesses. However... you could ;-)**

Why we Initially Chose SPIKE

ERNW Living Security.

- **Includes “proven” fuzzing strings**
- **Written in C**
- **Efficiency:**
  - **Write a generic program once (e.g. for TCP, UDP or Layer 2)**
  - **Add context-based payloads to this generic program via scripting interface (protocol descriptions)**
- **Very easy to use framework functions**
  - **Can be used in the scripts or in a “common C program”**
- **Complete code under GPLv2**

A new kid of town: Sulley

ERNW Living Security.

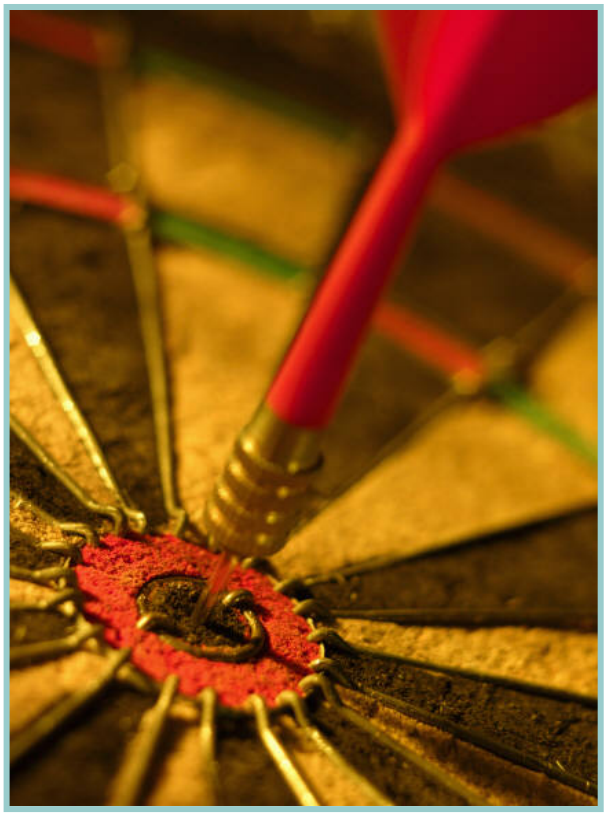
- **We decided to switch from SPIKE to the Sulley fuzzing framework**
  - **It can use SPIKE-Scripts without major changes**
  - **No more crappy SPIKE Parser ;)**
  - **Real python instead**

  - **NO MORE BYTE LIMITATION, because Sulley brings the s_bit_field which is _really_ useful for layer2 fuzzing**

Bring Sulley to layer2

ERNW Living Security.

- Very easy to implement
  - Sulley code is easy to modify
  - The patch only has some 100 lines
- We found (and fixed) a bug in the s_bit_field function, too.
- Changed the s_checksum implementation to build the IP-header checksum, for example.
- Touched the mutation algorithm to get a better coverage of the ‘packet space’.
- Additionally we add a flag to the s_size function to avoid the byte limitation.



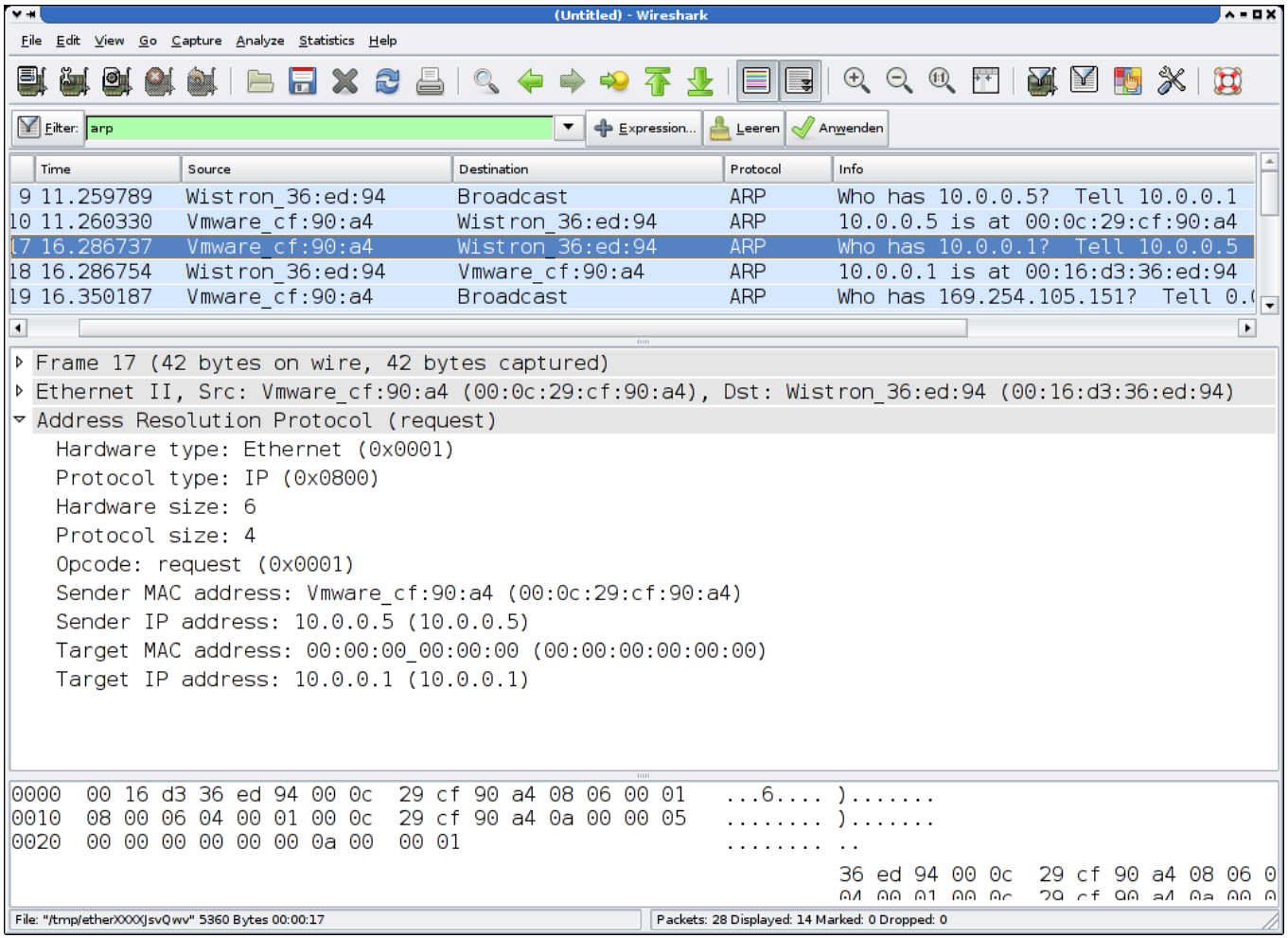
Protocol Definitions – The Simple Approach

ERNW Living Security.

- **Sniff packets**
- **Transform structures to prot. definition**
- ***Wireshark* is your friend here ;-)**

- **You still need a basic understanding of the stuff...**

Simple Example: ARP



Simple Example: ARP

ERNW Living Security.

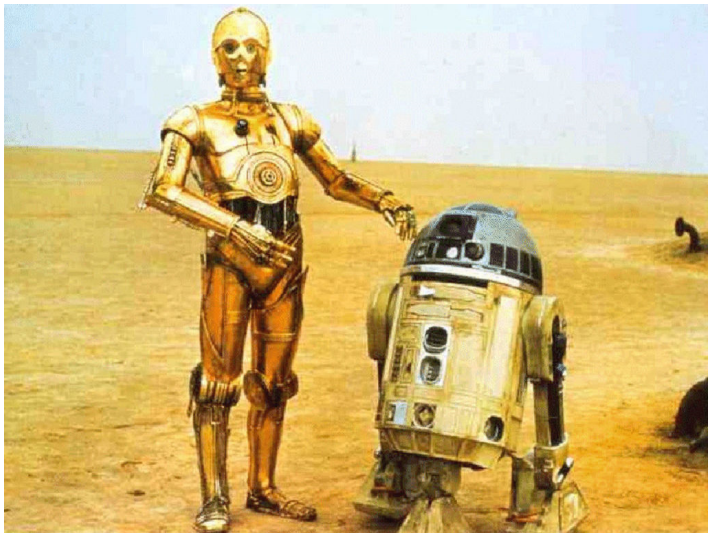
```
s_binary("0xff ff ff ff ff ff")
s_binary("0x01 02 03 04 05 06")
s_binary("0x08 06")

s_binary("0x00 01") #/* Hardware Type -> here Ethernet (1)*/
s_binary("0x08 00") #/* Protocol Type -> here IP (8) */
s_binary("0x06") #/* Hardware size -> here MAC (48Bit /6Byte) */
s_binary("0x04") #/* Protocol Size -> here IP (32Bit /4Byte) */
s_binary("0x00 01") #/* Opcode (1->request, 2->reply) */
s_binary("0x01 02 03 04 05 06") #/* MAC-Src */
s_binary("0xc0 a8 5f b5") #/* IP-Src */
s_binary("0x00 00 00 00 00 00") #/* MAC-Dst */
s_binary("0xc0 a8 5f b6") #/* IP-Dst */
s_random(0x0000, 1, 5)
```

Simple Example: ARP

ERNW Living Security.

```
# python arp.py
[12:41.24] current fuzz path:  -> arp
[12:41.24] fuzzed 0 of 25 total cases
[12:41.24] fuzzing 1 of 25
[12:41.24] xmitting: [1.1]
[12:41.24] fuzzing 2 of 25
[12:41.24] xmitting: [1.2]

[...]

[12:41.24] fuzzing 25 of 25
[12:41.24] xmitting: [1.25]
[12:41.24] all possible mutations for current fuzz node exhausted
```

Let‘s go practical then



Some of the protocol definitions we will have a look at:

- **MPLS**
- **VTP**
- **DTP**
- **EDP**
- **WLCCP**
- **LLDP**

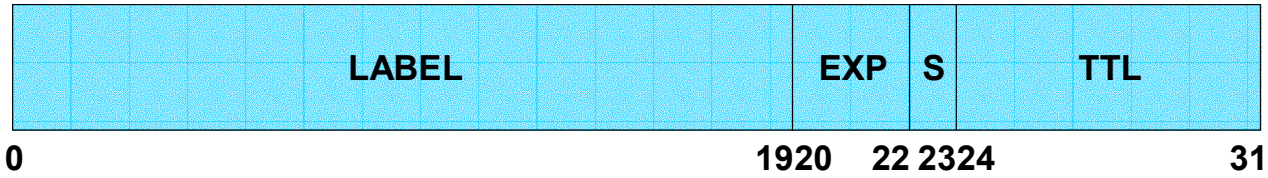
MPLS

ERNW Living Security.

- **Not really “a protocol” but a set of technologies and protocols.**
- **In the very basic technology a 32-bit header is inserted between Layer2 and Layer3 header (here on ethernet).**
- **Definition and subsequent fuzzing of these 32 bit are easy.**

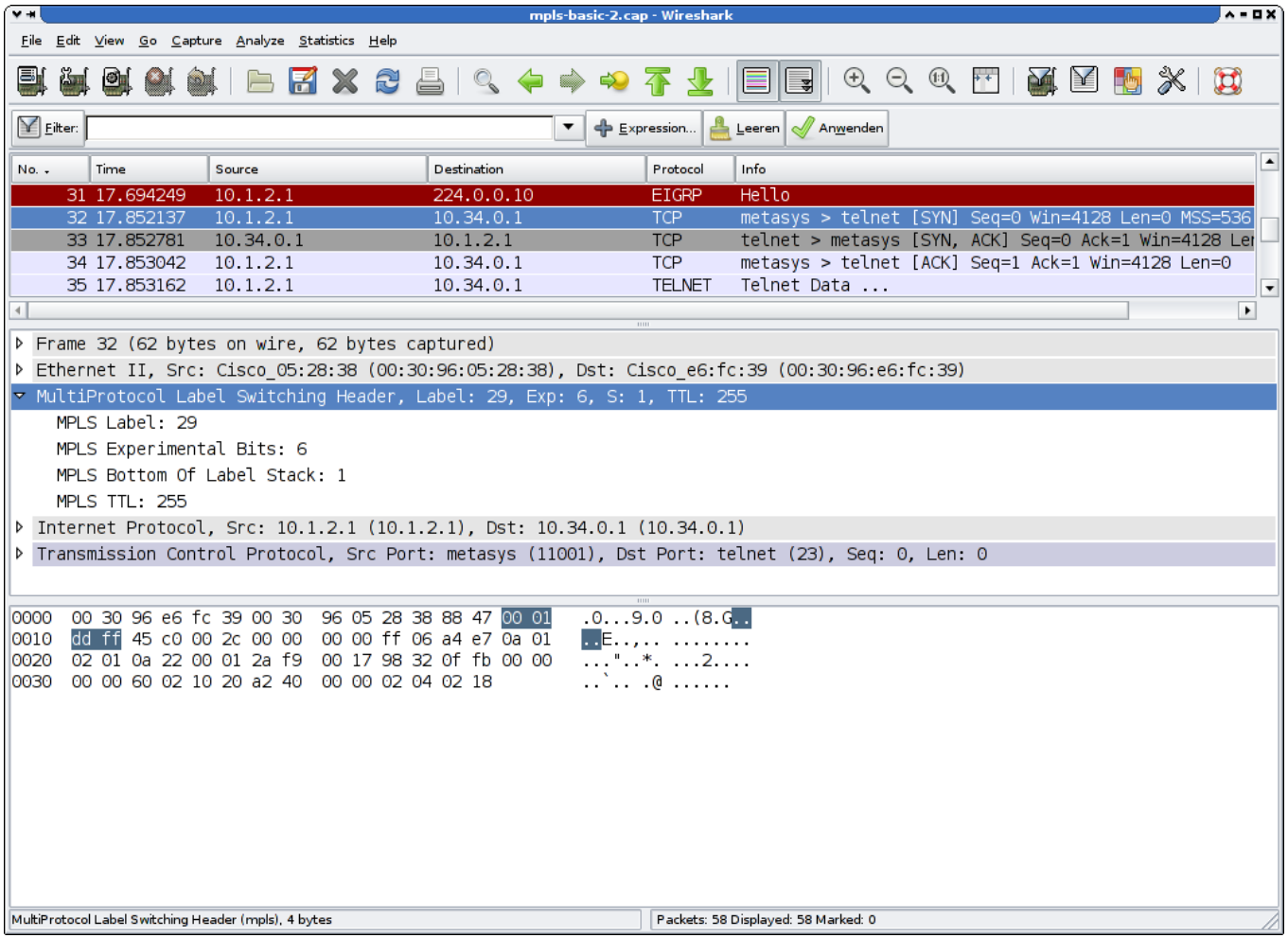
- **We did not split up the 32 bits into dynamic and static pieces (like the EXP part) or limit ranges.**
- **Testbed: some *Cisco* 7200 routers running *Service Provider* images. Processed packets without problems.**

MPLS Label Header



- 20-Bit Label
  - Short information entity without further internal structure
- 3-Bit Experimental-Bits (e.g. for CoS)
- 1-Bit Bottom-of-Stack Indicator (Label Stack)
- 8-Bit TTL-Field (Loop Mitigation)

MPLS (header) protocol definition



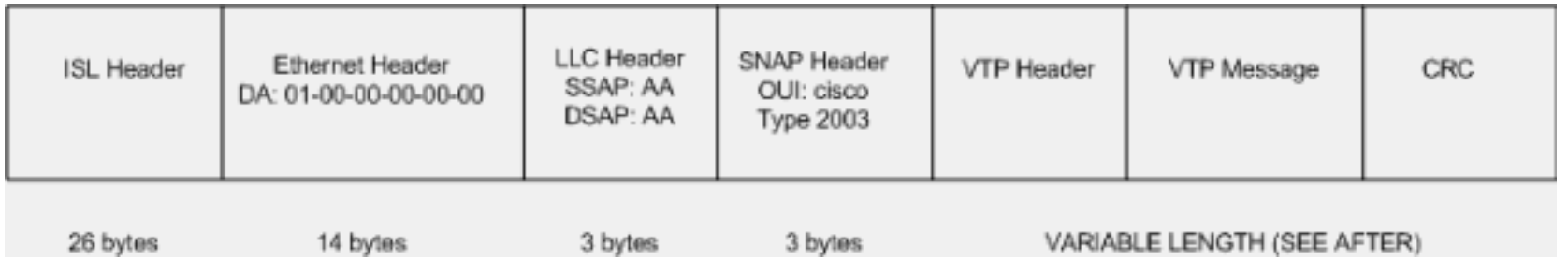
MPLS (header) protocol definition

```
s_binary("0x000dbc7e6e44")      #ETH Src
s_binary("0x014096ffffc0")      #ETH Dst
s_binary("0x8847")              #ETH Type

s_dword(0x0001ddff)             #MPLS Label

#Data
s_binary("0x45c0002c00000000ff06a4e70a0102010a22
  00012af9001798320ffb00000000600210 20a240000002
  040218")
```

VTP

ERNW Living Security.

- Good *Cisco* dokumentation
  - http://www.cisco.com/warp/public/473/21.html
- ISL or IEEE 802.1q encapsulated
- IEEE 802.3 Ethernet Header
- Logical Link Control Header
- Subnetwork Access Protocol Header



VTP packet format

ERNW Living Security.

- **3 types of VTP messages:**
  - **Summary Advertisements**
  - **Subset Advertisements**
  - **Advertisement Requests**

VTP packet format

ERNW Living Security.

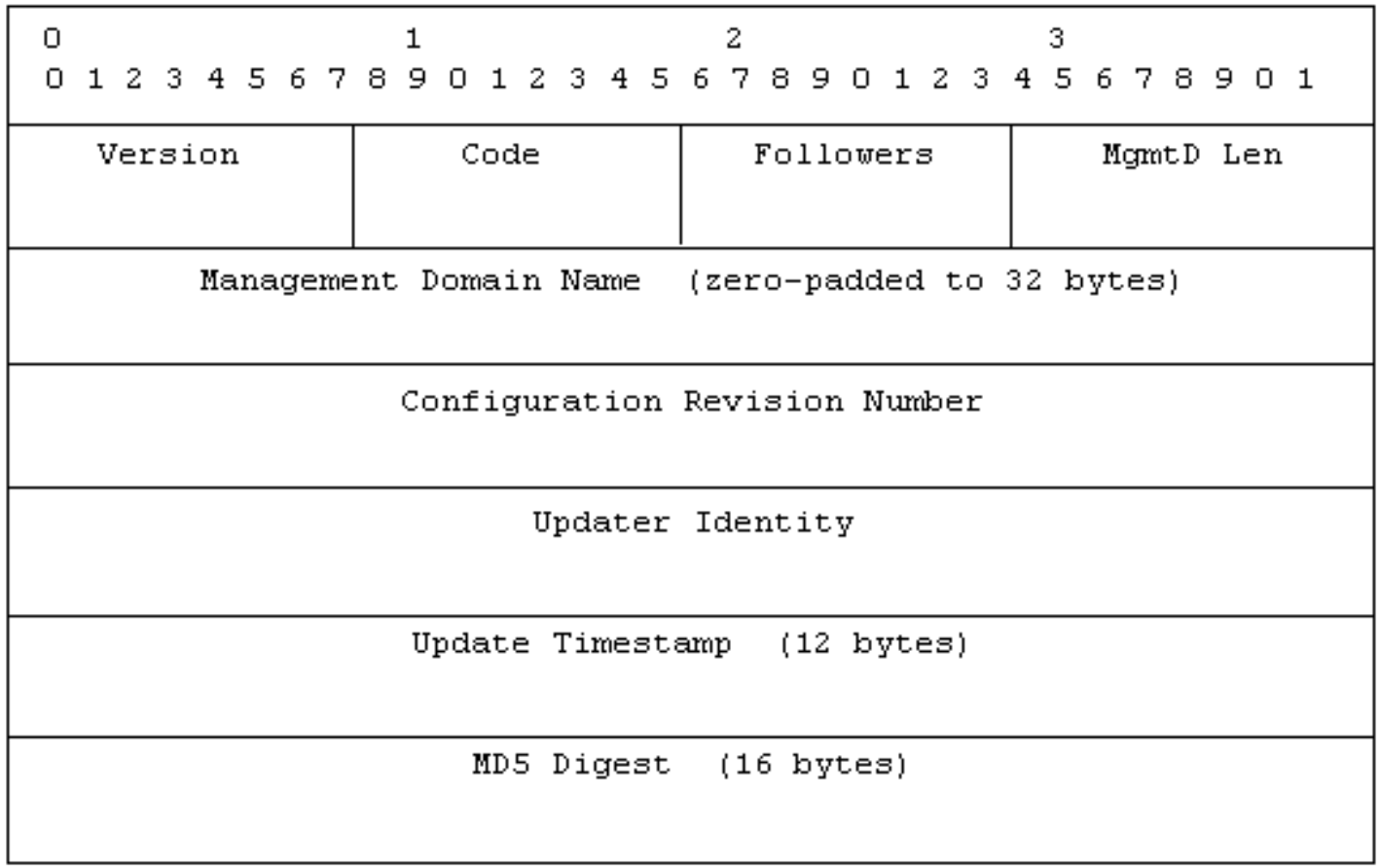
- **Summary Advertisement Packets**

- **(Per default) transmitted every five minutes**
- **Include the name of the *VTP domain***
- **Populate the current revision number of the VLAN-database**

VTP packet format

ERNW Living Security.

Summary Advert Packet Format:



VTP packet format

ERNW Living Security.

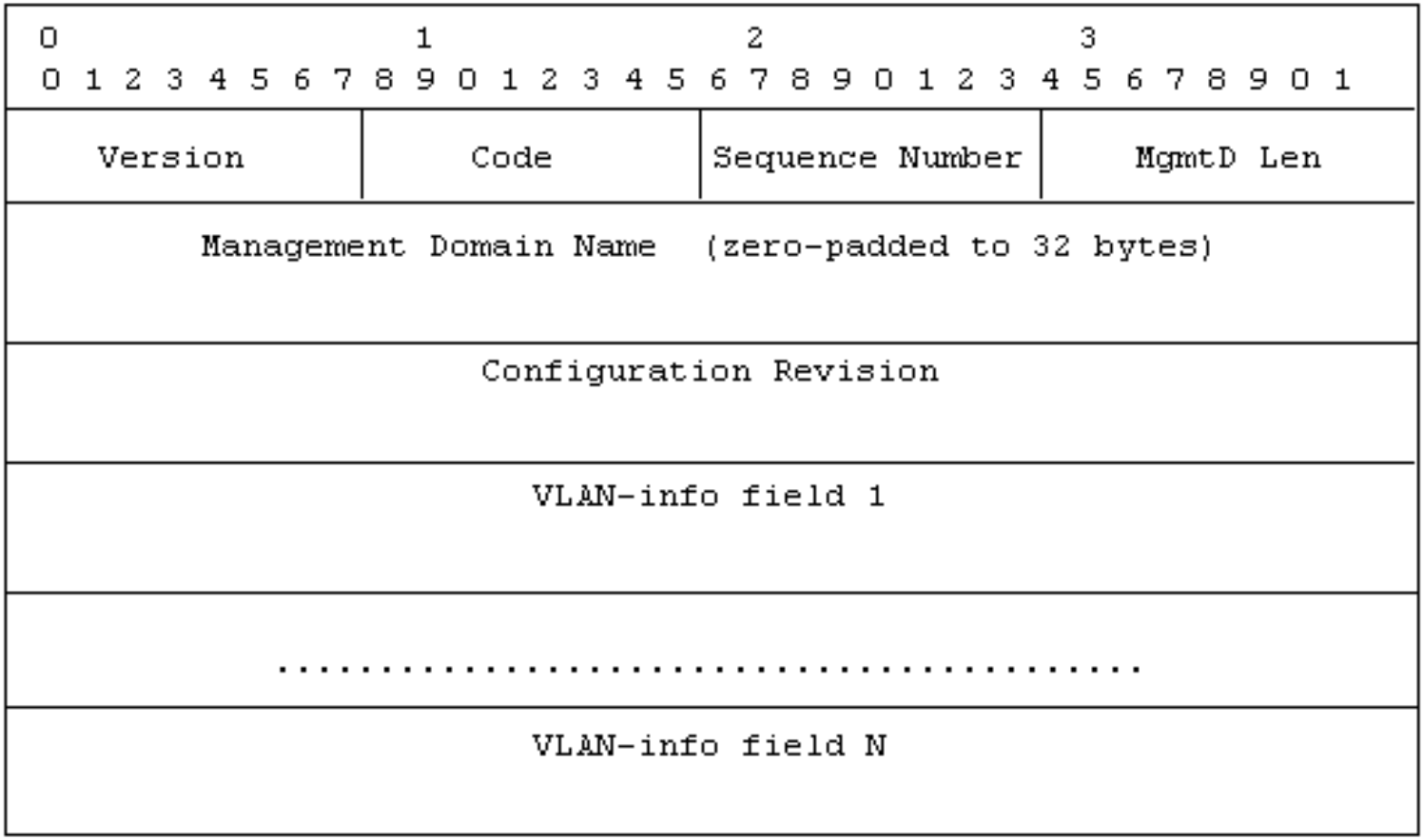
- **Subset Advertisement Packet**

- **Transmitted in answer to an advertisement request**
- **Contains multiple VLAN-Info fields**
- **One or more *Subset Advertisement* packets represent the complete VLAN-Database**

VTP packet format

ERNW Living Security.

Subset Advert Packet Format:



VTP packet format

ERNW Living Security.

- **Advertisement request Packets**

- **Transmitted in three cases:**
  - **VLAN-Database is empty (after reset)**
  - **VTP-Domain changed**
  - **Summary Advertisement with higher revison no. received**

Spike scripts
VTP Summary Advertisement

ERNW Living Security.

```
s_block_start("802.3")
s_binary("0x01 00 0c cc cc cc")
s_binary("0x00 01 02 03 04 05")
s_size("802.3", length=2, inclusive=True, endian=">")

s_binary("0xaa") #/* DSAP */
s_binary("0xaa") #/* SSAP */
s_binary("0x03") #/* func */
s_binary("0x00000c") #/* Orga-code */
s_binary("0x2003") #/* VTP */

s_byte(1) #/* version */
s_binary("0x01") #/* code */
s_byte(0) #/* followers */
s_size("MgmtD", length=1) #/* MgmtD length */
s_block_start("MgmtD")
s_binary("0x66757a7a696e67") #/* Mgmt Domain = "fuzzing" */
s_block_end("MgmtD") #/* end MgmtD length */
s_binary("0x00000000000000000000000000000000000000000000000000000000") #/* fill Domain up to 32 byte
    */
s_dword(111) #/* configuration revision number - 4byte */
s_dword(0) #/* update identity - 4byte */
s_bit_field(0, 96) #/* update timestamp - 12bytes */
s_binary("0x0000000000000000") #/* md5 digest / password - 16 bytes length */

s_block_end("802.3")
```

Spike scripts
VTP Subset Request

ERNW Living Security.

```
s_block_start("802.3")
s_binary("0x01 00 0c cc cc cc")
s_binary("0x00 01 02 03 04 05")
s_size("802.3", length=2, inclusive=True, endian=">")

s_binary("0xaa") #/* DSAP */
s_binary("0xaa") #/* SSAP */
s_binary("0x03") #/* func */
s_binary("0x00000c") #/* Orga-code */
s_binary("0x2003") #/* VTP */

s_byte(1)#/* version */
s_binary("0x03") #/* code */
s_byte(0, 3)       #/* rsvd */
s_size("MgmtD", length=1) #/* MgmtD length */
s_block_start("MgmtD")
s_binary("0x66757a7a696e67") #/* Mgmt Domain = "fuzzing" */
s_block_end("MgmtD") #/* end MgmtD length */
s_binary("0x00000000000000000000000000000000000000000000000000000000") #/* fill Domain to 32
   byte */
s_dword(0)         #/* start value (4byte) */

s_block_end("802.3")
```

VTP, our Results

ERNW Living Security.

- **Tested with several Cisco switches (29xx, 35xx, 3750, 6509).**

- **Nearly no effect :(**
- **[albeit packets obviously processed]**

Possible cause for VTP (non-)results

ERNW Living Security.

DTP Packet Format

ERNW Living Security.

- **No *Cisco* documentation publicly available**

- **But there is a *wireshark* parser...**
- **Which saved us a lot of work ;-)**
- **Looking at the *yersinia* code would have been another option...**

DTP Packet Format

- **Same encapsulation as VTP with the Subnetwork Access Protocol Header type of 0x2004**
- **Based on Type-Length-Value entries with:**
  - **2 Bytes type**
  - **1 Byte length**
  - **The data**

- **4 known types:**
  - **Domain – contains the DTP Domain name**
  - **Status – contains the DTP Status**
  - **Type – contains the DTP Type**
  - **Neighbor – contains the MAC address of the neighbor**

Sulley scripts – DTP

```
s_block_start("Domain")
s_binary("0x0001")                                          #/* Type: Domain */
s_size("Domain", length=2, inclusive=False, endian=">")     #/* Domain length */
s_binary("0x46555a5a494e4720202020202020202020202020202020202020202020202000")
 #/* Domain: "Fuzzing" */
s_block_end("Domain")
s_block_start("Status")
s_binary("0x0002")                                          #/* Type: Status */
s_size("Status", length=2, inclusive=False, endian=">")     #/* Status length */
s_byte(0x03)                                                #/* Status */
s_block_end("Status")
s_block_start("DTPtype")
s_binary("0x0003")                                          #/* Type: DTPtype */
s_size("DTPtype", length=2, inclusive=False, endian=">")    #/* DTPtype length */
s_byte(0xa5)                                                #/* DTPtype */
s_block_end("DTPtype")
s_block_start("Neighbor")
s_binary("0x0004")                                          #/* Type: Neighbor */
s_size("Neighbor", length=2, inclusive=False, endian=">")   #/* Neighbor length */
s_bit_field(0x0c7ce846d595, 48)                             #/* Neighbor MAC-Adress */
s_block_end("Neighbor")
```

Results – DTP

- **Tested against same testbed.**
- **On some devices/images while fuzzing (on one switchport) strange things happen:**
  - **Trunk on other (!!) ports goes down and up and down up …**
  - **Some ports set to mode *blocking***
  - **The device blinks like a Christmas tree**
  - **…**

This does _not_ look good ;-)

ERNW Living Security.

```
00:57:55: FEC: get-fechannel: port (Fa0/2) not part of fechannel line
   = 2311 func = strata_dma_done_desc_rx: Received packet for unit 0,
   swport 0
Inst base port = 0, dcb port = 0
[0000]: {01000CCCCCCC} {000102030405} 002E AAAA
00:57:55: 00100300 000C 2004 0001 0400 0002 0400 0003
00:57:55: 00200401 0004 0000 0000 0000 0000 0000 0000
00:57:55: 00300000 0000 000B 6C61 6C61 6C61
00:57:55: line = 746 func = process_rx_packet iport = 0x0
linkType = 114 line = 879 func = process_rx_packet
line = 2207 function= strata_dma_done_desc_rx
[ … SNIP … ]
pm_vlan_rem_port: vlan 4093, port 1
pm_vlan_rem_port: vlan 4094, port 1
cled_vp_list_fwdchange: state 0(fwd 1)
cled_vp_list_fwdchange: [1] blocked 1

hmat_handle_pm_vp_fwdchange Interface Fa0/2, Vlan 1 changed state to
   blocking
mat_enable_disable_addrs: type:2, port:Fa0/2
```

EDP Packet Format

ERNW Living Security.

- **Proprietary *Extreme Networks* Protocol**
  - **Used in the same scenario as LLDP or CDP.**
  - **It is also used to transfer ESRP information. So ESRP is a special TLV type of EDP.**
  - **No public documentation available, but again, read the Wireshark-Source ☺ even if not the hole protocol is covered …**

EDP Packet Format

- **IEEE encapsulation with Destination 00:e0:2b:00:00:00**
- **Type-Length-Value entries with:**
  - **1 Byte TLV-Marker: 0x99**
  - **1 Byte type**
  - **2 Bytes length**
  - **The data**
- **5 types:**

Null	**0x00**	**–**	**last Type in every Packet**
Display	**0x01**	**–**	**Name of the Device**
Info	**0x02**	**–**	**System Information**
VLAN-Info	**0x05**	**–**	**VLAN-Information**
ESRP	**0x08**	**–**	**…**

Sulley scripts – EDP (excerpt)

```
[…]
s_word(0x0000)                                  #/* Machine ID Type */
s_bit_field(0x000130fe84f3, 6*8)                        #/* Machine MAC */

s_block_start("Info")
s_binary("0x99")                                        #/* TLV Marker */
s_binary("0x02")                                        #/* TLV Type: Info */
s_size("Info", length=2, inclusive=True, endian=">")        #/* TLV-Length */
s_word(0x0007)                                          #/* Slot */
s_word(0x003b)                                          #/* Port */
s_word(0x0000)                                          #/* Virt chassis */
s_bit_field(0x000000, 6*8)                              #/* Reserved */
s_dword(0x0b020204)                                     #/* Version */
s_bit_field(0x00000000000000000000000000000000, 16*8)           #/* Connections */
s_block_end("Info")
s_block_start("Display")
s_binary("0x99")                                        #/* TLV Marker */
s_binary("0x01")                                        #/* TLV-Type: Display */
[…]
```

Results – EDP

ERNW Living Security.

- **Tested against a Summit48si**
- **While / After fuzzing …**
  - **… the device didn't send EDP anymore**
  - **… displaying the EDP-Information didn't work in desirable way, instead an “Address load Exception“ is returned ;-)**
  - **… configuration changes may crash the hole system**

Results – EDP

```
Address load Exception
Exception Program Counter: 0x803530bc
Status Register: 0x34008d01
Cause Register: 0x00001010
Access Address : 0x41414145
Task: 0x81d42830 "tConsole"
 00x800fc328 shellExec      +5f8:  execShowEdp ( 81d3d078 ,  0 ,  80230e18 ,  0 )
 00x80116290 execShowEdp    +a8 :  cliShowEdpPort ( 81d3d078 ,  81d3cc10 ,  0 ,  0 )
 00x802b225c cliShowEdpPort +10c:  p2BtreeFirst ( 808288f8 ,  0 ,  0 ,  f3 )
```

Results – EDP (the same, but direct in the output)

```
[…]
Remote-system: HD000002~Y (Version 4.2.2)
                Remote-ID=00:00:5e:55:55:55:55:55
                Remote-Port=1793:15105   Age=14
        Remote-system:
    AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
    AAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAddress load ExceptionAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAException Program Counter: 0xAAAAAAAAAAA803530bcAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAStatus Register: 0xA34008d01AAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAACause Register: 0xAA0000AAAAAAAAAAAAAAAA1010AAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAccess Address : 0xA41414145AAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAATask: AAAAAAAAAAAAAAA0xAAAAAAAAAAAAAAAAA8429e700AAAAAAAAAAAA
    "AAAAAAAAAAAAAAAAAAtEdpTaskAAAAAAAAAAAA"
 00x8076806c vxTaskEntry    +c   :  edpTask ( 0 ,  0 ,  0 ,  0 )AAAAAAAAAAAAAAAAA
 00x802af050 edpTask        +160:  edpTimerExpiration ( eeeeeeee ,  eeeeeeee ,  eeeeeeee ,
    eeeeeeee )AAAAAAAA
 00x802affb4 edpTimerExpiration+48 :  edpAgeNeighbor ( 8429e088 ,  eeeeeeee ,  eeeeeeee ,
    eeeeeeee )AAAAAAAAA
 00x802b1ec8 edpAgeNeighbor +88 :  edpAgeVlanInfo ( 8081b378 ,  81655bd8 ,  3 ,  1 )AAAAAAAAAA
 00x802b1cd0 edpAgeVlanInfo +2c :  p2BtreeFirst ( eeeeeeee ,  eeeeeeee ,  eeeeeeee ,  eeeeeeee
    )AAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
    AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA)
                Remote-ID=00:00:f3:84:fe:30:01:00
                Remote-Port=16706:16706   Age
[…]
```

Another protocol definition: WLCCP

ERNW Living Security.

- **The next protocol on our list was Cisco's proprietary *Wireless Lan Context Control Protocol***
- **Serves for some special (wire based) Inter-AP communication in Cisco networks**
- **We think protocol is flawed (architecture wise) anyway. Might be topic for another talk ;-)**

- **No documentation available**
- **Wireshark gives a starting point, but as the implementation seems incomplete and flawed (at least at layer2) there was (and is) a lot more work to do.**

The WLCCP Sulley script (excerpt ;-)

ERNW Living Security.

```
from sulley import *

s_initialize("WLCCPoUDP")

s_block_start("Payload")
s_byte(0x1c)                                    #Version
s_bit_field(1, 2)                               #SAP Version
s_bit_field(0, 6)                               #SAP ID
s_word(0x0008)                                  #Dest Node type
s_size("Payload", length=2, endian=">")         #Length
s_bit_field(0, 2)                               #Subtype
s_bit_field(11, 6)                              #Base MsgType
s_byte(0x00)                                    #Hops
s_byte(0x0001)                                  #MsgID
s_bit_field(8192, 16)                           #Flags
s_word(0x0001)                                  #Originator Node type
s_bit_field(0x000cce333225, 48)                 #Orginator MAC
s_word(0x0008)                                  #Responder Node type
s_bit_field(0x000dbc7e6e44, 48)                 #Responder MAC
s_word(0x0001)                                  #Requestor Node type
s_bit_field(0x000cce333225, 48)                 #Requestor MAC
s_byte(0x00)                                    #AAA MsgType
s_byte(0x04)                                    #AAA AuthType
s_byte(0x00)                                    #AAA KeyMgmtType
s_byte(0x00)                                    #Status
s_bit_field(0, 44, fuzzable=False)              #Fill up with zero
[…]
```

Results – WLCCP

ERNW Living Security.

- **Not too many (reliable) results, probably because WLCCP requires quite ”some state”**
- **However every now and then APs crash and need hard resets afterwards. So far we are not able to reproduce this behavior in a controlled manner**
- **Next steps:**
  - Reverse engineer the protocol
  - Understand the WLCCP state machine and build different scripts for all the states

Short return to spike_l2 – LLDP

ERNW Living Security.

- **While fuzzing LLDP with spike_l2 we figured out some stuff we wont keep back**
- **Recently we found a bug in CISCOs LLDP-Implementation**
- **The script causing this isn't implemented for sulley yet**

Short return to spike_l2 – LLDP

ERNW Living Security.

- **Pretty complex protocol**
- **Works with *Type-Length-Value* (TLV) structures**
- **Ethernet-Header (type 0x88cc), packets sent to multicast-address** `01:80:c2:00:00:0e`
- **Due to "SPIKE's byte limitation" (and odd TLVs) initially it was not possible to fuzz LLDP, with SPIKE and L2-addon**
- **=> addition of *s_binary_type_and_block_size_lldp()***
  - **gets an integer as the TLV-type**
  - **Plus char* as the name of the block**

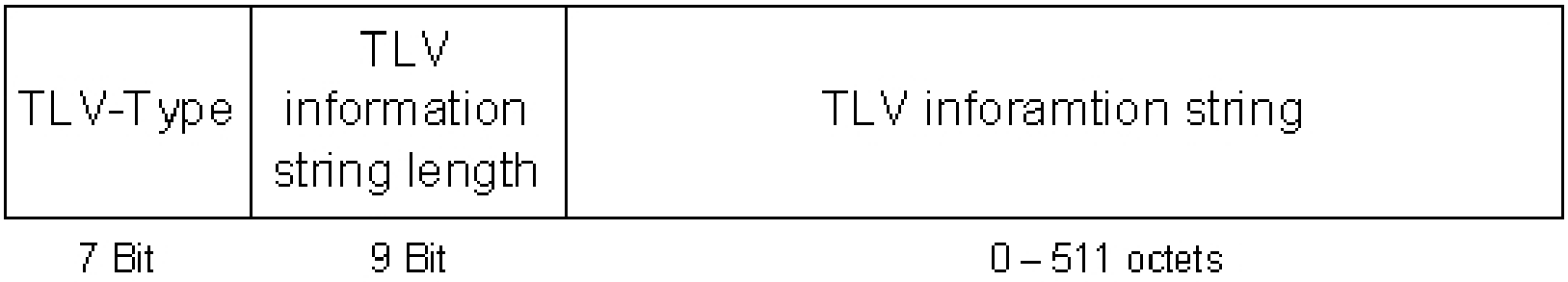
Short return to spike_l2 – LLDP

ERNW Living Security.

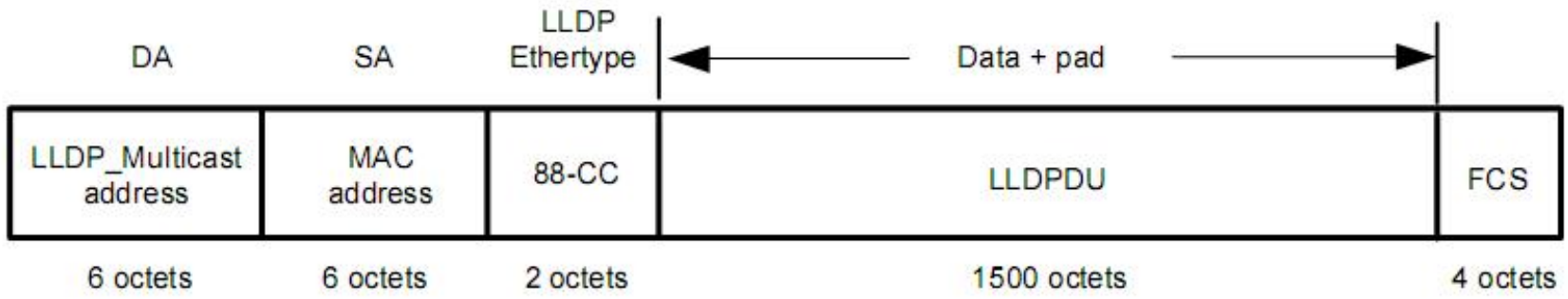
- **When multiple packets (containing different information) arrive from same source MAC address the packets are discarded**

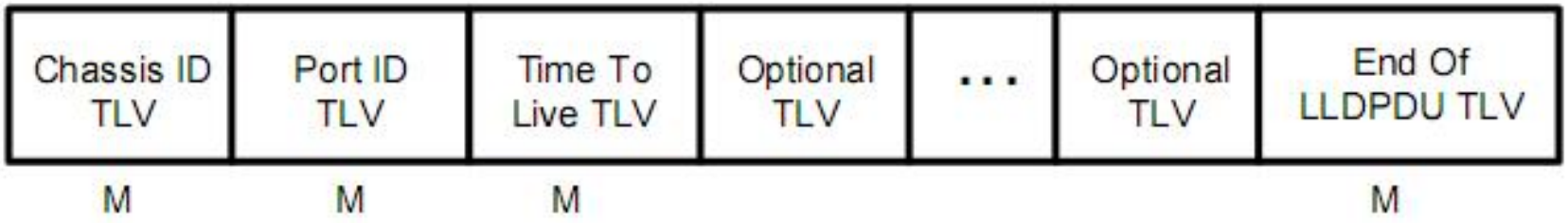
  => random source MACs needed
  => *generic_send_l2 rewritten* with *random_mac_option*

LLDP format


TLV-Type
TLV information string length
TLV inforamtion string
7 Bit
9 Bit
0 – 511 octets

LLDP format (2)





M - mandatory TLV - required for all LLDPDUs

LLDP (small excerpt!)

ERNW Living Security.

```
s_binary_type_and_block_size_lldp(1, "block_chassis");
 /* TLV Type: Chassis Id(1) + TLV Length:7 */
s_block_start("block_chassis");
s_push_int(7, 3);                              /* Chassis Id Subtype: 1,2,3,4,5,6 or 7 */
s_string_variable_sized("000130f9ada0", 1, 255);
 /* Chassis Id (dependes on Chassis ID Subtype) */
s_block_end("block_chassis");

s_binary_type_and_block_size_lldp(2, "block_port");
 /* TLV Type: Port Id (2) + TLV Length: 4 */
s_block_start("block_port");
s_int_variable(7, 3);                   /* Port Id Subtype:1,2,3,4,5,6 or 7 */
s_string_variable_sized("312f31", 1, 255);              /* Port Id: 1/1 */
s_block_end("block_port");

s_binary_type_and_block_size_lldp(3, "block_ttl");
 /* TLV Type: Time to Live (3) + TLV Length:2 */
s_block_start("block_ttl");
s_push_int(120,5);                     /* Seconds: 120 */
s_block_end("block_ttl");

s_binary("00 00");                     /* TLV Type: End of LLDPDU (0) + TLV Length: 0 */
```

Results – LLDP

```
02:29:33: LLDP rx state on FastEthernet0/3 set to WAIT FOR FRAME
02:29:33: LLDP advertisement packet RX'd on intf FastEthernet0/3
02:29:33: LLDP advertisement packet RX'd on intf FastEthernet0/3
02:29:33: LLDP rx state on FastEthernet0/3 set to RX FRAME
02:29:33: LLDP unknown tlv type 127 recd - ignoring it
02:29:33: LLDP malformed optional TLV 127 found - ignored
02:29:33: LLDP entry update - new neighbor C:\ discovered
[…]
02:29:33: LLDP-MED orig state on FastEthernet0/3 is DOWN, rcvd caps 0x0000
02:29:33: LLDP rx state on FastEthernet0/3 set to WAIT FOR FRAME
02:29:33: LLDP malformed optional TLV 127 found - ignored
02:29:33: LLDP entry update - new neighbor  discovered
02:29:33: LLDP-MED orig state on FastEthernet0/3 is DOWN, rcvd caps 0x0000
02:29:33: LLDP rx state on FastEthernet0/3 set to WAIT FOR FRAME
02:29:33: LLDP rx state on FastEthernet0/3 set to RX FRAME
02:29:33: LLDP unknown tlv type 127 recd - ignoring it
02:29:33: LLDP malformed optional TLV 127 found - ignored
02:29:33: LLDP entry update - new neighbor
../../../../../../../../../../../../localstart.asp%00 discovered
02:29:33: LLDP-MED orig state on FastEthernet0/3 is DOWN, rcvd caps 0x0000
02:29:33: LLDP rx state on FastEthernet0/3 set to WAIT FOR FRAME
```

Results (reproducible) – LLDP

```
c3560#more flash:crashinfo/crashinfo_1
Cisco IOS Software, C3560 Software (C3560-ADVIPSERVICESK9-M), Version
12.2(40)SE, RELEASE SOFTWARE (fc3)
Copyright (c) 1986-2007 by Cisco Systems, Inc.
Compiled Fri 24-Aug-07 01:43 by myl

Instruction TLB Miss Exception (0x1200)!

SRR0 = 0x2A2A2A28  SRR1 = 0x00029230  SRR2 = 0x0059574C  SRR3 = 0x00021200
ESR = 0x00000000  DEAR = 0x00000000  TSR = 0x8C000000  DBSR = 0x00000000

CPU Register Context:
Vector = 0x00001200  PC = 0x2A2A2A28  MSR = 0x00029230  CR = 0x40000002
LR = 0x2A2A2A2A  CTR = 0x00000000  XER = 0x0000003F
R0 = 0x2A2A2A2A  R1 = 0x02F44E28  R2 = 0x00000000  R3 = 0x02F45050
R4 = 0x019CFC7D  R5 = 0xFFFFFFFF  R6 = 0x02F44D90  R7 = 0x00000000
R8 = 0x00000000  R9 = 0x02F450B3  R10 = 0x02F450B3  R11 = 0x02F450B2
[...]
Stack trace:
PC = 0x2A2A2A28, SP = 0x02F44E28
Frame 00: SP = 0x2A2A2A2A    PC = 0x2A2A2A2A
```

Other protocols

ERNW Living Security.

- **We got a lot of other protocol definitions ready, but not jet tested.**
- **To give you a short overview:**
  - lwapp, pvstp, udld, cdp, stp, vrrp …
- **As testing is the most time consuming part, were happy for every helping hand.**

The Code

ERNW Living Security.

- **Get it from our website:**
  - http://www.ernw.de/download/l2spike.tar.bz2
  - http://www.ernw.de/download/l2sulley.tar.bz2
- **Given these are stress testing tools ;-), no problems to expect with §202c...**
- **We will continue developing this stuff and will add new protocol definitions (there are so many interesting L2 protocols out there...)**

Summary

ERNW Living Security.

- **SPIKE did a good job, Sulley will do even better.**
- **We learned a lot about fuzzing frameworks and protocols during that project.**
- **Hopefully you find some of the project‘s outcome helpful...**
- **And, btw: some network devices from $SOME_BIG_VENDOR might have parser problems, too…**

Talking about code… some old stuff updated: *snmpattack.pl*

ERNW Living Security.

```
usage: snmpattck.pl [-hIrv] [-A type] [-c comm1,comm2] [-C tftp] [-f target] [-s type]
[-l delimiter] {ip/range | input file}

-A type   : Do APC specific attacks (type: 1 = allON, 3 = allOFF, 4 = allREBOOT)
-c comm   : Add communities to check for (comma separated)
-C tftp   : Do Cisco specific attacks and specify a tftp server for config upload
-f target : Switch to flood-mode
-h        : Print this help
-I        : Do InnoMedia specific attacks
-l        : Parse IPs from file, seperatet with the given delimiter
-p port   : The port for tcp syn scan (default = 80)
-r        : Test for RO / RW community
-s type   : Scans the given ip/range (type: snmp, icmp, syn | default = snmp)
-t num    : Count of parallel scans (default = 10)
-v        : Be verbose

scan and attack all found devices:
# $0 -I 10.0.0.0/24

scan and use all founds as relay hosts:
# $0 -s syn -p 21 -v -f 1.2.3.4 10.0.0.0/24
```

- http://www.ernw.de/download/snmpattack.pl

Questions?

ERNW
Living Security.

Thanks for your attention!

ERNW
Living Security.