Advanced Protocol Fuzzing – What We Learned when Bringing Layer2 Logic to *SPIKE Land*

... and later we went on to *Sulley Land* ;-)

Enno Rey & Daniel Mende
{erey,dmende}@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

- **The Need for a Layer2 Fuzzer**
- **Fuzzing Landscape & Options**
- **Why we Initially Chose SPIKE**
- **Limitations & Additional Features we Implemented**
- **Some Protocols and Results**
- **The Journey Goes on... here comes *Sulley***

Definition

- **“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*

The Need for a Layer 2 Fuzzer

- **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?

- **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 ;-)**

Fuzzing Landscape & Options

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

Why we Initially Chose SPIKE

- **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**
- **In the meantime we prefer Sulley… wait for later part of talk…**

How to run SPIKE

- **Get package**
- **Unpack, ./configure, make**

- **Open shell and use one**
  - **of the programs for specific purposes**
    - **probably a script is also needed**
  - **of the more generic programs**
    - **you have the write your own script(s) per protocol**

- **Or write a new specific / generic program (we did)**

SPIKE, Sample Script

```
//netbios
s_int_variable(0x81,3); //session type //sessionon request
s_int_variable(0x00,3); //flags
s_binary_block_size_halfword_bigendian_variable("netbiosbl
  ock");

s_block_start("netbiosblock");
//*SMBSERVER
s_string_variable(" CKFDENECFDEFFCFGEFFCCACACACACACA");
s_binary("00");
//LOCALHOST
s_string_variable(" EMEPEDEBEMEIEPFDFECACACACACACAAA");
s_binary("00");
s_block_end("netbiosblock");
```

Protocol Definitions – The Simple Approach

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

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

Simple Example: ARP

```
s_binary("00 01"); /* Hardware Type -> here Ethernet (1)*/
s_binary("08 00"); /* Protocol Type -> here IP (8) */
s_binary("06"); /* Hardware size -> here MAC (48Bit) */
s_binary("04"); /* Protocol Size -> here IP (32Bit) */
s_binary("00 01"); /* Opcode (1->request, 2->reply) */
s_string_variable("01 02 03 04 05 06"); /* MAC-Src */
s_string_variable("c0 a8 5f b5"); /* IP-Src */
s_string_variable("00 00 00 00 00 00"); /* MAC-Dst */
s_string_variable("c0 a8 5f b6"); /* IP-Dst */
```

Problem here:
s_string_variable takes any string, not just those with length of six bytes

=> We added a new function *s_string_variable_sized*

General Limitations

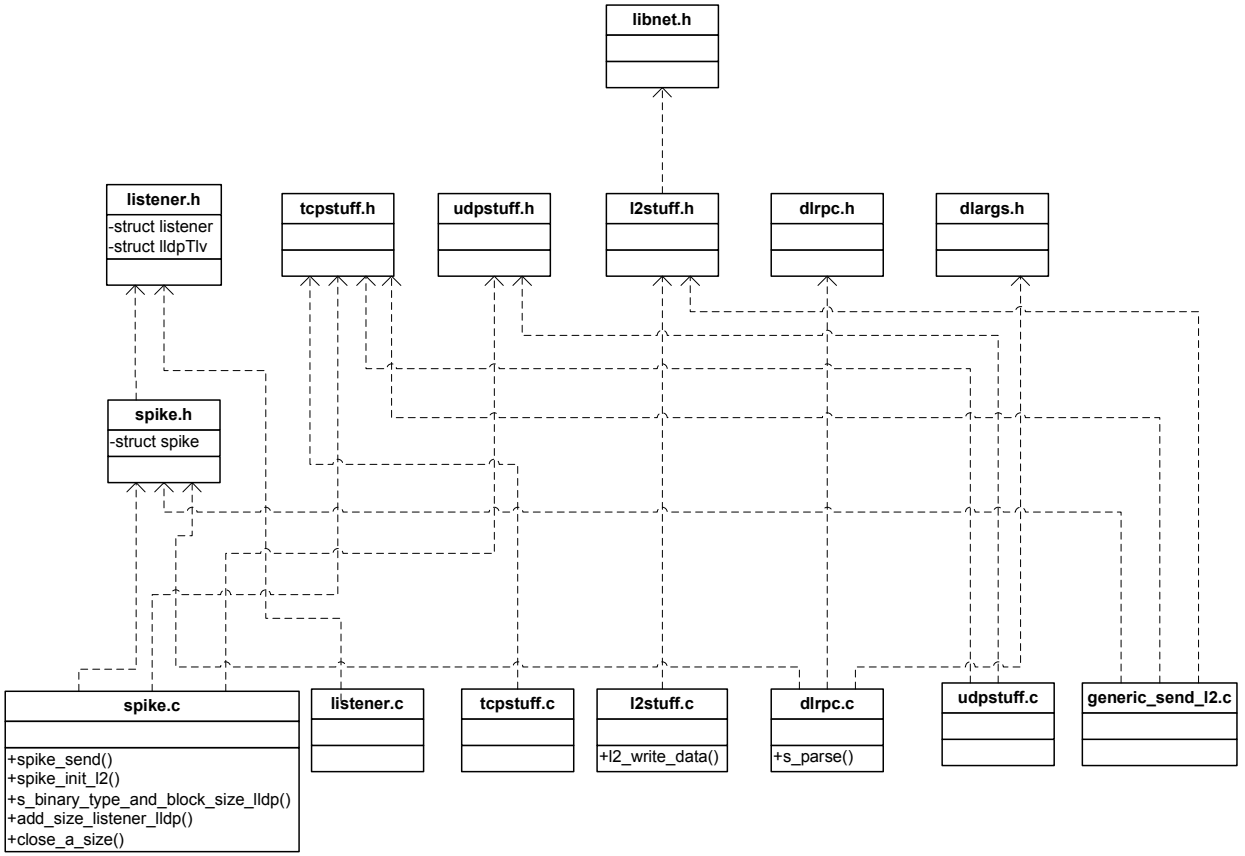
- **SPIKE mostly does string / integer based fuzzing**
  - **=> addition of *s_string_variable_sized()***
- **SPIKE is byte-oriented**
  - No handling of protocol information with “odd sizes” possible
- **No handling of bit fields (e.g. TLVs)**
  - **One of the reasons why we later switched to Sulley**
- **No fuzzing with/of predefined values possible**
  - Added function *s_binary_selection*
  - Did not work from SPK scripts due to parser weaknesses

Additional Features we Implemented

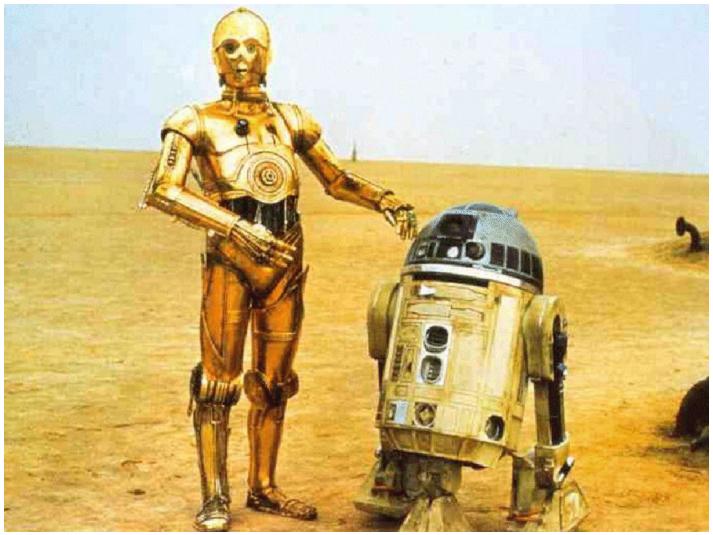
- **Generic L2 sender (Ethernet II and IEEE802.3)**
  - **Selection of random or fixed ethernet-src**

- **Additional functions**
  - **s_random_fuzz(), s_random_fuzz_repeat() fuzz completely random data with fixed size [based on POSIX rand()]**
  - **s_binary_type_and_block_size_lldp()**
  - **l2_write_data()**
  - **s_binary_selection()**
  - **s_string_variable_sized()**

Overview ;-)


libnet.h
listener.h
-struct listener
-struct lldpTlv
tcpstuff.h
udpstuff.h
l2stuff.h
dlrpc.h
dlargs.h
spike.h
-struct spike
spike.c
+spike_send()
+spike_init_l2()
+s_binary_type_and_block_size_lldp()
+add_size_listener_lldp()
+close_a_size()
listener.c
tcpstuff.c
l2stuff.c
+l2_write_data()
dlrpc.c
+s_parse()
udpstuff.c
generic_send_l2.c

Let‘s go practical then



Some of the protocol definitions we‘ve added so far:

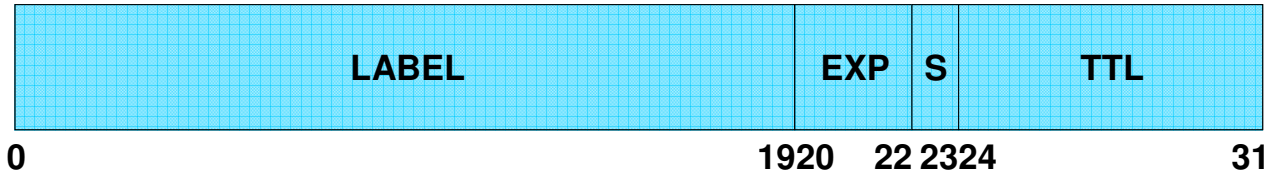
- **MPLS**
- **LLDP**
- **VTP**
- **DTP**
- **WLCCP (only for Sulley)**

MPLS

- **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

- **Uses INTELENDIANWORDs (= 32 Bits)**

```
is_int_fuzz_variable(9);   /* 9 equivalent to INTELENDIANWORD */
s_binary( "PACKET CONTENT" );
...
```

- **Demo**

LLDP

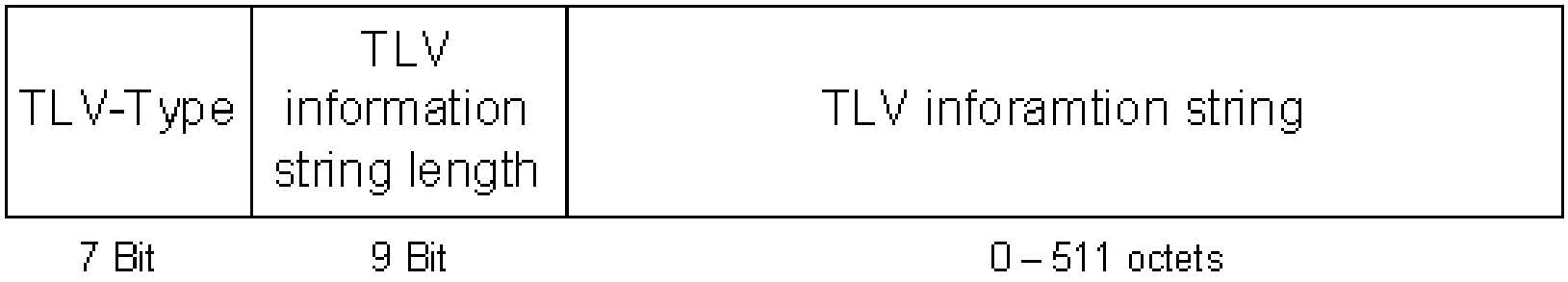
- **Pretty complex protocol**
- **Works with *Type-Length-Value* (TLV) structures**
- **Ethernet-Header (type 0x88cc), packets sent to multicast-address `01:80:c2:00:00:0e`**
- **Due to "SPIKE's byte limitation" (and odd TLVs) initially it was not possible to fuzz LLDP, with SPIKE and L2-addon**
- **=>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

LLDP (2)

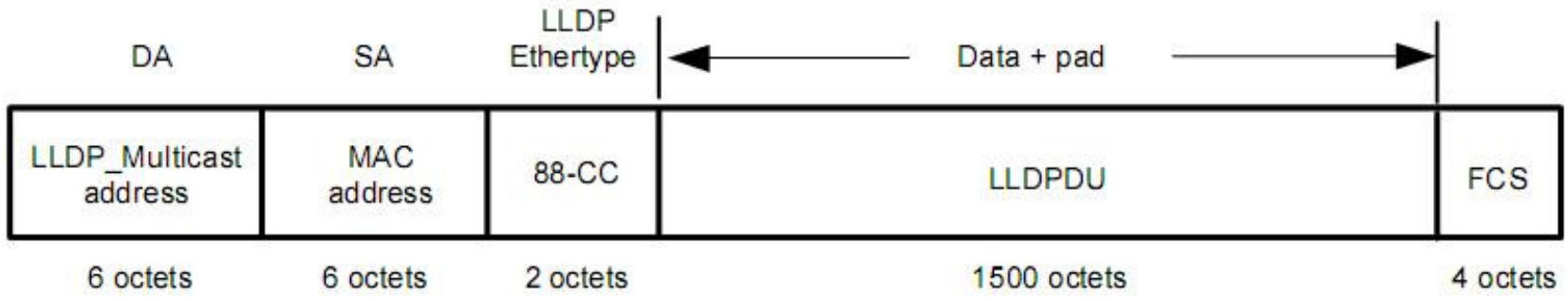
- **When multiple packets (containing different information) arrive from same source MAC address the packets are discarded**

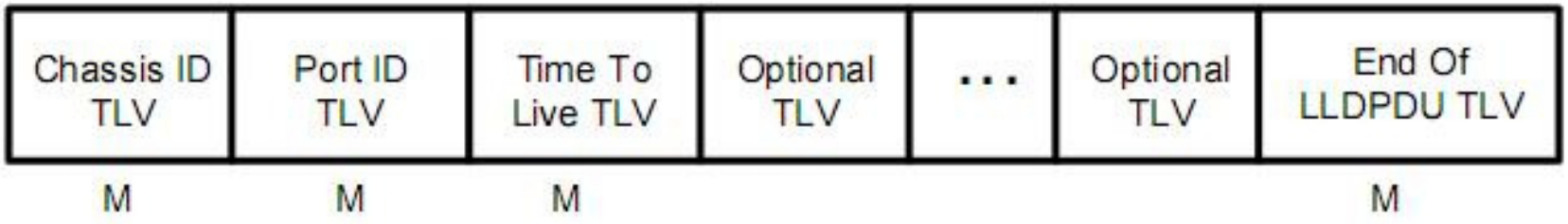
  => random source MACs needed
  => *generic_send_l2 rewritten* with *random_mac_option*

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!)

```
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

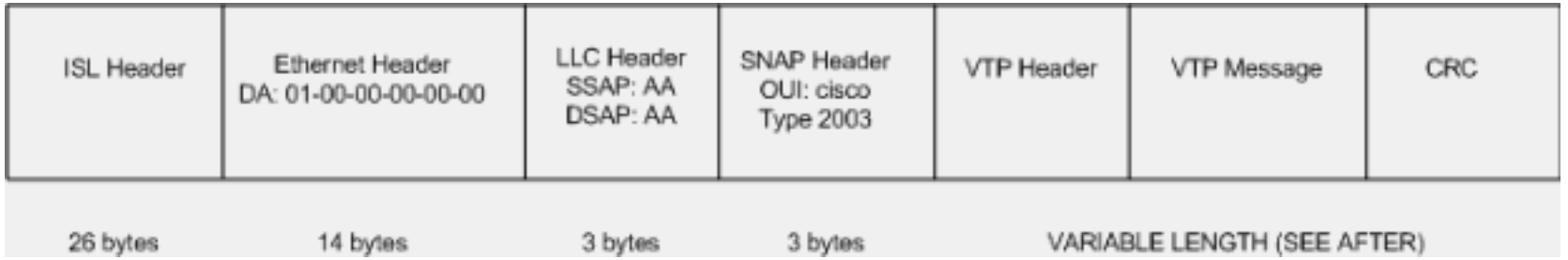
```
02:29:33: LLDP rx state on FastEthernet0/3 set to WAIT FOR FRAME
02:29:33: LLDP advertisement packet RX'd on intf FastEthernet0/3
02:29:33: LLDP advertisement packet RX'd on intf FastEthernet0/3
02:29:33: LLDP rx state on FastEthernet0/3 set to RX FRAME
02:29:33: LLDP unknown tlv type 127 recd - 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
```

VTP

- **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

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

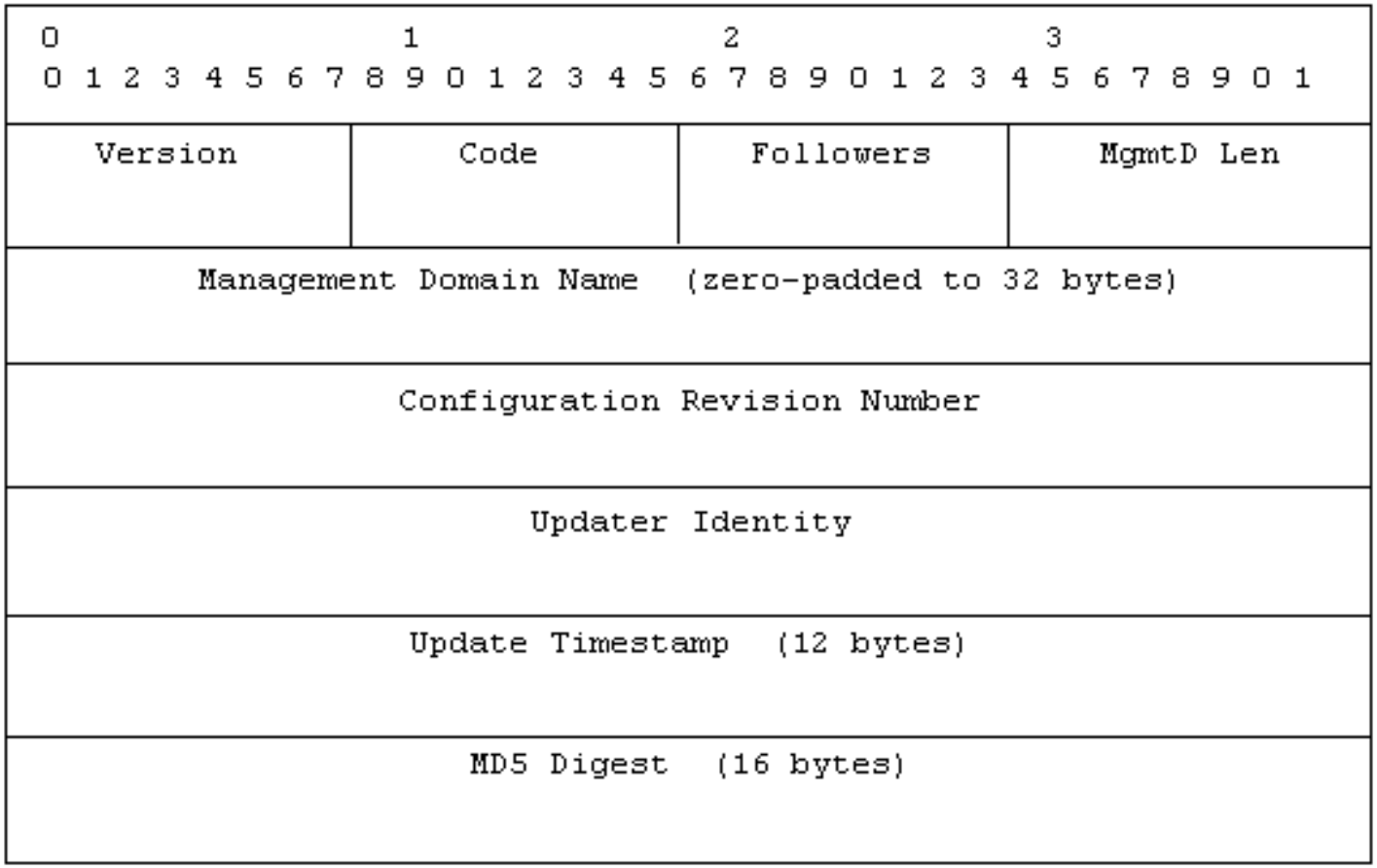
VTP packet format

- **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

Summary Advert Packet Format:



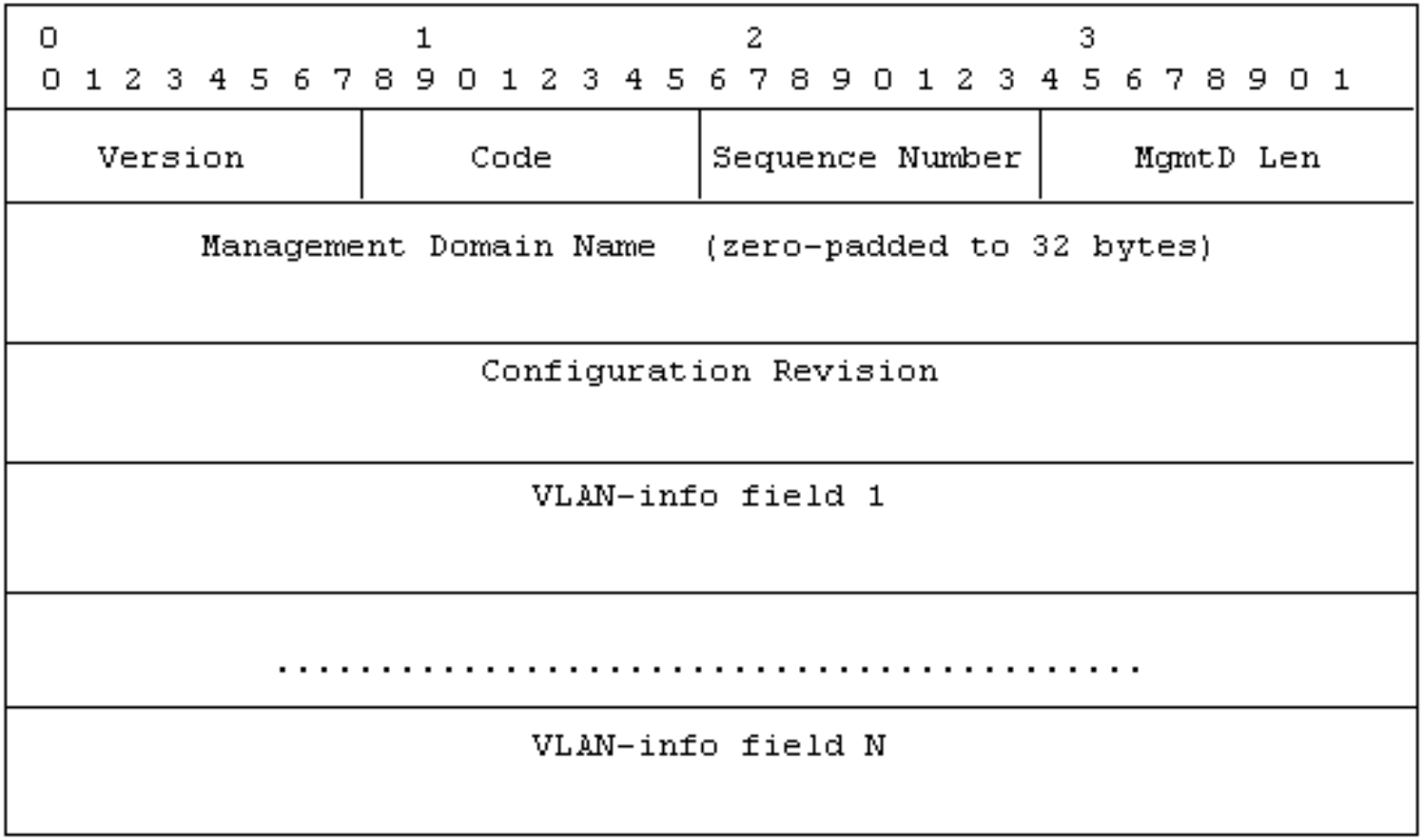
VTP packet format

- **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

Subset Advert Packet Format:



VTP packet format

- **Advertisement request Packets**

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

Spike scripts
VTP Summary Advertisement

```
s_binary("aa"); /* DSAP */
s_binary("aa"); /* SSAP */
s_binary("03"); /* func */
s_binary("00000c"); /* Orga-code */
s_binary("2003"); /* VTP */

s_int_variable(1,3); /* version - ONEBYTE */
s_binary("01"); /* code */
s_int_variable(0,3); /* followers - ONEBYTE */
s_binary_block_size_byte_variable("MgmtD"); /* MgmtD length */
s_block_start("MgmtD");
s_binary("66757a7a696e67"); /* Mgmt Domain = "fuzzing" */
s_block_end("MgmtD"); /* end MgmtD length */
s_binary("00000000000000000000000000000000000000000000000000"); /* fill Domain to
   32 byte */
s_int_variable(111,1); /* configuration revision number - BINARYBIGENDIAN */
s_int_variable(0,1); /* update identity - BINARYBIGENDIAN */
s_random_fuzz(12); /* update timestamp */
s_binary("00000000000000000"); /* md5 digest / password - 16 bytes length */
```

Spike scripts
VTP Subset Request

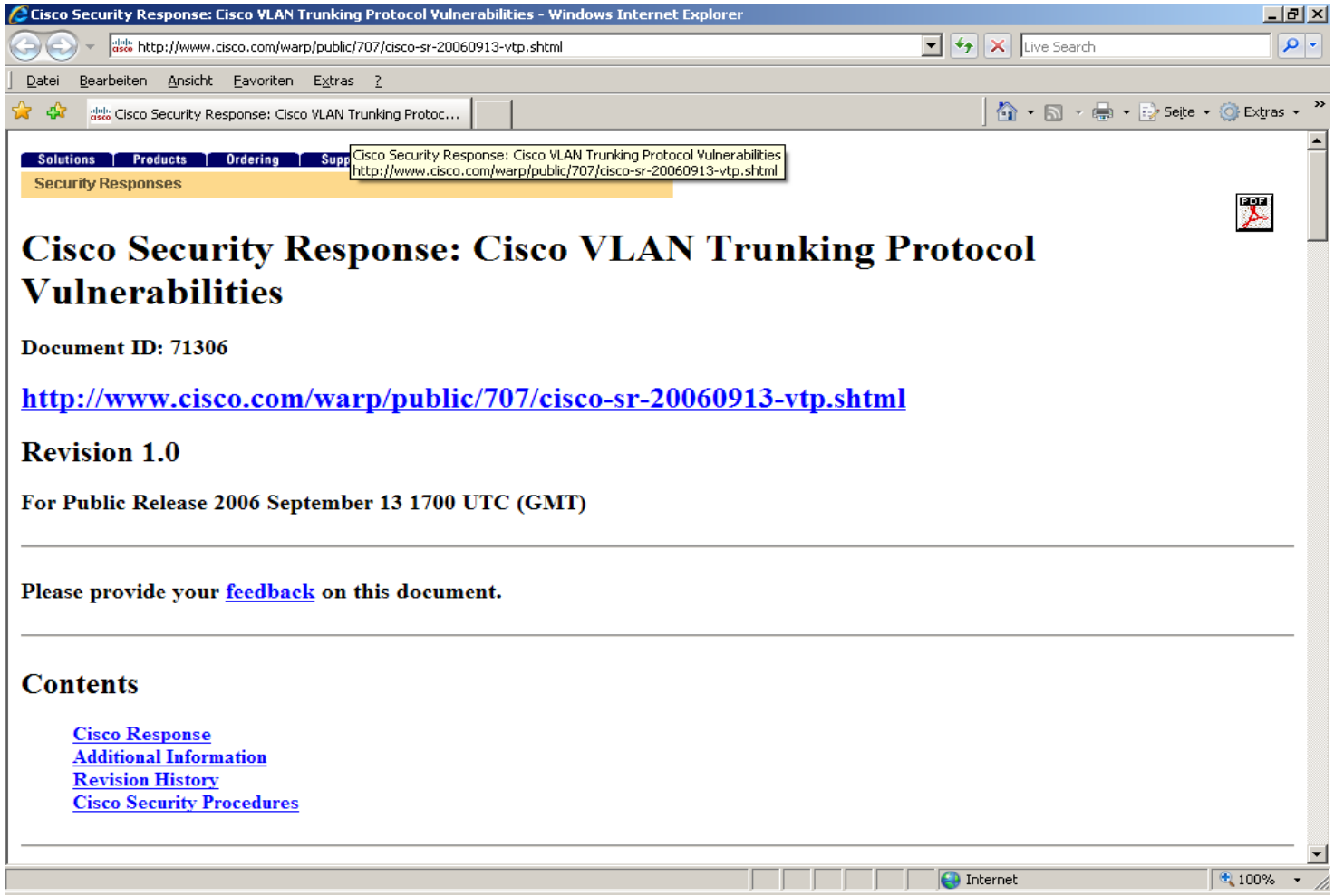
```
s_binary("aa"); /* DSAP */
s_binary("aa"); /* SSAP */
s_binary("03"); /* func */
s_binary("00000c"); /* Orga-code */
s_binary("2003"); /* VTP */

s_int_variable(1, 3);       /* version - ONEBYTE */
s_binary("03"); /* code */
s_int_variable(0, 3);       /* rsvd - ONEBYTE */
s_binary_block_size_byte_variable("MgmtD"); /* MgmtD length */
s_block_start("MgmtD");
s_binary("66757a7a696e67"); /* Mgmt Domain = "fuzzing" */
s_block_end("MgmtD"); /* end MgmtD length */
s_binary("00000000000000000000000000000000000000000000");
   /* fill Domain to 32 byte */
s_random_fuzz(32);  /* start value */
```

VTP, Results

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

- **Nearly no effect ☹**
  [albeit packets obviously processed]

Possible cause for VTP (non-)results



DTP Packet Format

- **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**

Changes made to Spike

- **Modified the layer2stuff to support IEEE 802.3 headers**
- **Modified the creation of fuzz-integers to cover the whole WORD range**

- **And of course: created a Spike script for DTP**

Spike scripts – DTP

```
s_binary("aa"); /* DSAP */
s_binary("aa"); /* SSAP */
s_binary("03"); /* func */
s_binary("00000c"); /* Orga-code */
s_binary("2004"); /* DTP */

s_block_start("Domain");
s_binary("0001"); /* Type: Domain */
s_binary_block_size_byte("Domain"); /* Domain length */
s_binary("00"); /* Domain: none */
s_block_end("Domain");

s_block_start("Status");
s_binary("0002"); /* Type: Status */
s_binary_block_size_byte("Status"); /* Status length */
s_int_variable(0, 3);   /* Status - ONEBYTE */
s_block_end("Status");

s_block_start("DTPtype");
s_binary("0003"); /* Type: DTPtype */
s_binary_block_size_byte("DTPtype"); /* DTPtype length */
s_int_variable(1, 3);   /* DTPtype - ONEBYTE */
s_block_end("DTPtype");

s_block_start("Neighbor");
s_binary("0004"); /* Type: Neighbor */
s_binary_block_size_byte("DTPtype"); /* Neighbor length */
s_int_variable(0, 1);   /* Neighbor byte 0,1 - BINARYBIGENDIAN */
s_int_variable(0, 1);   /* Neighbor byte 2,3 - BINARYBIGENDIAN */
s_int_variable(0, 1);   /* Neighbor byte 4,5 - BINARYBIGENDIAN */
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 ;-)

```
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
```

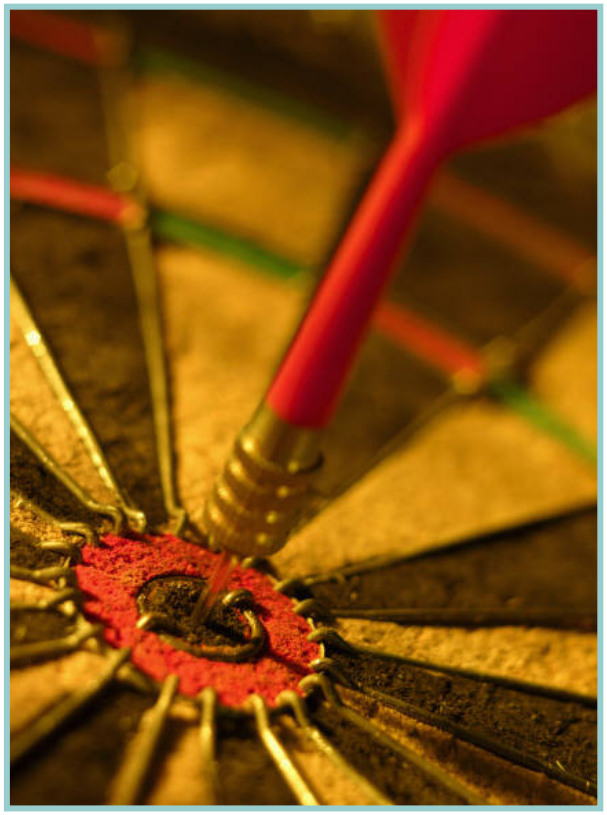
“Blinking like a Christmas tree“

A new kid in town: Sulley

- **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

- **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.**
- **Additionally we added a flag to the *s_size function* to avoid the byte limitation.**



First Sulley scripts

- **After bringing L2 logic to *Sulley* we tested the new capability with some of the SPIKE scripts**
  - ARP was very easy
    - Only adjust the syntax (from SPIKE to python)
    - Add some Sulley session handling stuff
  - DTP was easy, too. But we did not see the same results… why?
    - Other fuzz strings
    - We didn't fuzz the whole variable range, as we did in SPIKE

The Sulley ARP script

```
from sulley import *

s_initialize("arp")

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)

sess = sessions.session(proto="layer2", iface="eth0")
sess.connect(s_get("arp"))

target = sessions.target("layer2", 1234)
sess.add_target(target)

sess.fuzz()
```

Another protocol definition: WLCCP

- **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 ;-)

```
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
```

Results – WLCCP

- **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

The Code

- **Will this stuff be available?**
- **Yes! On 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...)**

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

```
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, separated 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

Summary

- **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...**
- **See you @ Saturday Night Party ;-)**

Questions?

Thanks for your attention!

www.troopers08.org