ERNW
Living Security.

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

Enno Rey & Simon Rich

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.**

Who we are

- **Enno:** **Old school networker and security geek**
  Founder of ERNW GmbH in 2001
- **Simon:** **Student & security researcher @ ERNW**
- **ERNW:** **Germany based security consultancy**
  Currently 12 FTEs + several students/stagiaires

Heavily engaged in security research which is usually conducted in teams with students.

Agenda

ERNW Living Security.

- **The Need for a Layer2 Fuzzer**
- **Fuzzing Landscape & Options**
- **Why we Chose SPIKE**
- **Introduction to Protocol Fuzzing with SPIKE**
- **Limitations & Additional Features we Implemented**
- **Some Protocols and Results**
- **Lessons Learned & Outlook**

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

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 ;-)**
- **One can learn a lot about protocols when building fuzzers..**

Fuzzing Landscape & Options

ERNW Living Security.

- **There are many fuzzers available**
- **Most of them: unmaintained or one-man projects**
- **Interesting Fuzzing Frameworks**
  - **Spike**
  - **autodafé**
  - **Peach**
  - **GPF – General Purpose Fuzzer**
    - **With Evolutionary Fuzzing System (EFS)**
- **Very promising new-kid-in-town: *Sulley* (one of the sponsors here will be delighted about this mention ;-) Seems to be a milestone and might mark the beginning of a new era of fuzzers/frameworks. Michael loves it ;-)**

Why we 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 either in a “common C program”**
- **Complete code under GPLv2**

Introduction to Protocol Fuzzing with SPIKE

ERNW Living Security.

- **Two types of fuzzing programs are possible with SPIKE**
  - **Specialized fuzzers for one protocol**
    - **Know your protocol and write a fuzzer for it**
    - **Used for stateful fuzzing**
  - **More generic fuzzers for wider use**
    - **Know the layer and the protocol which is in use**
    - **Write a generic program that can handle the payload over the layers beyond**
    - **e.g. generic_send_tcp, generic_send_udp**

How to run SPIKE

ERNW Living Security.

- **Get the package**
- **Unpack it, ./configure, make**
- **Open a shell and use one**
  - **of the programs for specific purposes**
    - **Possibly a script is also needed**
  - **of the more generic programs**
    - **Maybe you have the write your own script(s)**
- **Or write a new specific / generic program**

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

ERNW Living Security.

- **Sniff packets**
- **Copy structures to protocol 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()***
- **No handling of bit fields**
  - **“Problem” not only of the fuzzer ;)**

More on Limitations

- **SPIKE seems not to be able to work with fields with “well defined content“ (e.g. LLDP Chassis-ID).**
- **Only completely random content [s_string_variable] with subsequent *variable* size or fuzzing within defined “integer range“ [e.g. ls_int_fuzz_variable] possible**

- **=> new function s_binary_selection added**
  => currently can‘t be used from SPK script ... obviously our design of the function was not too well thought out ;-)
- **Can only be used directly in (C) code**
  => use only when needed

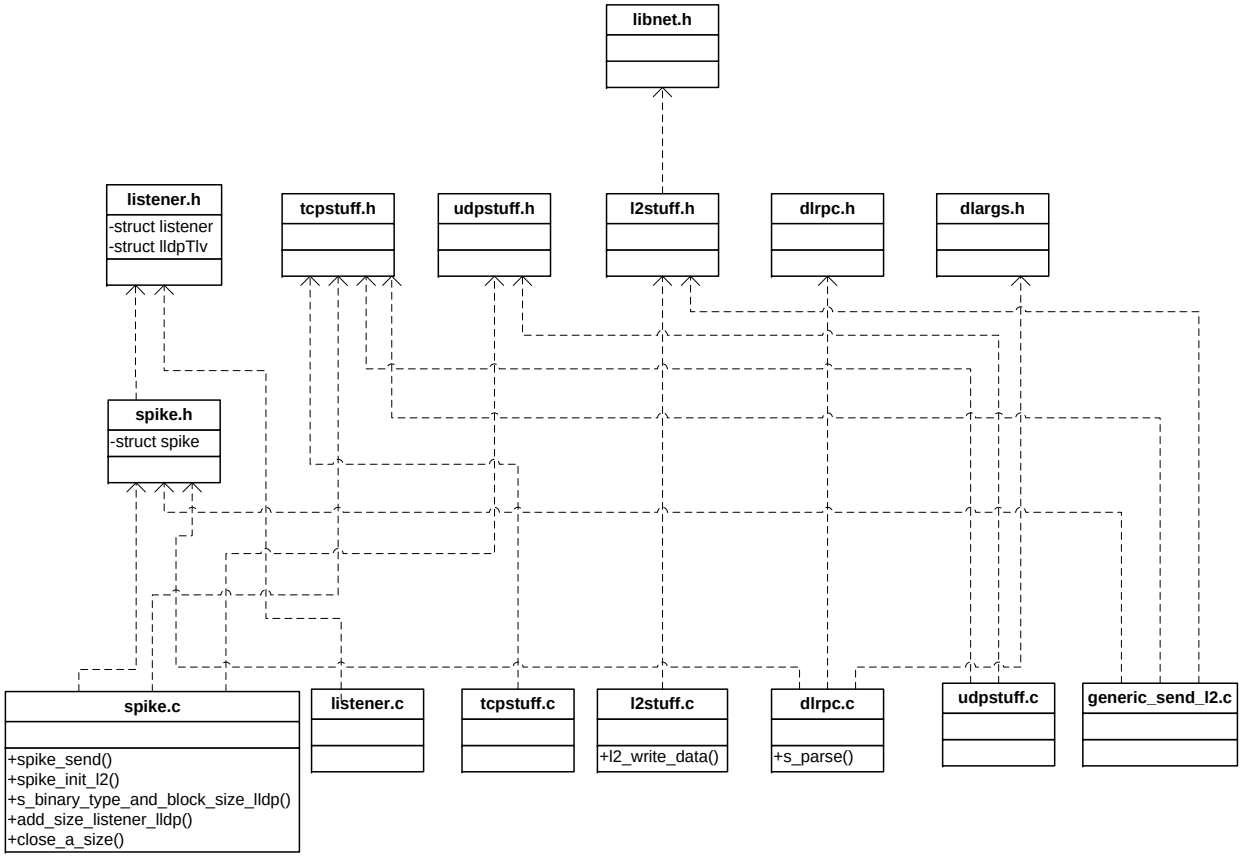
Additional Features we Implemented

ERNW Living Security.

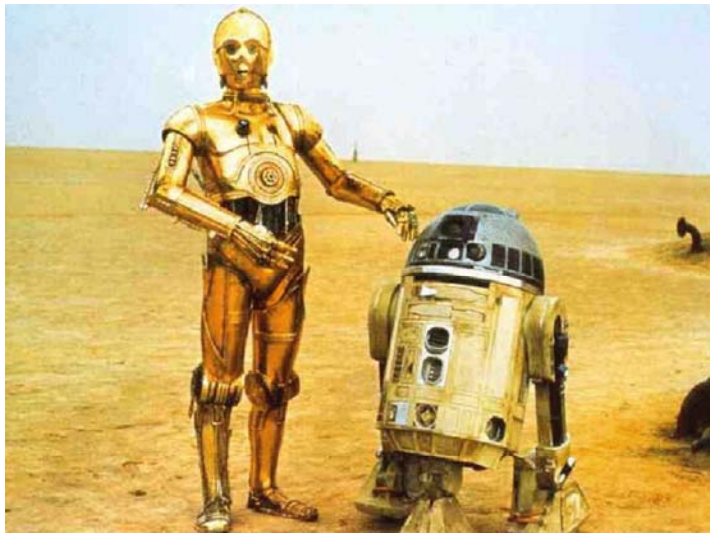
- **Generic L2 sender (Ethernet II and IEEE802.3)**
  - **Selection of random or fixed ethernet-src**

- **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**

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

- **In use: INTELENDIANWORD**

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

- **Demo**

LLDP

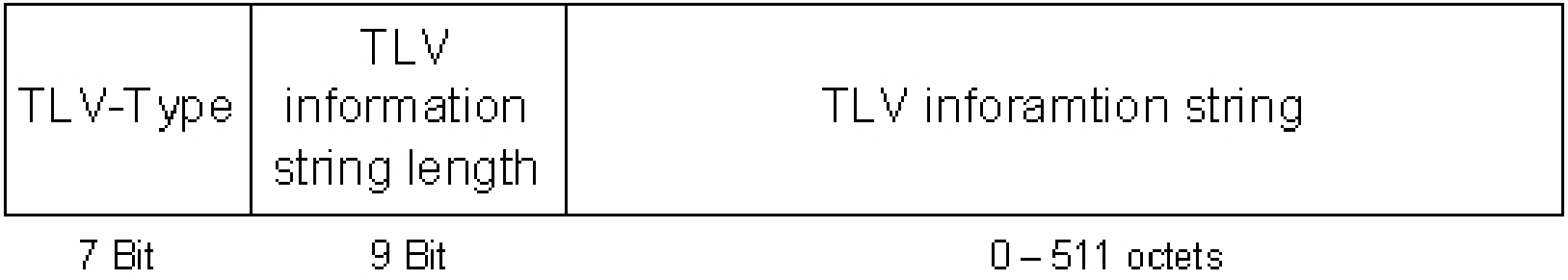
- **Pretty complex protocol**
- **Works with *Type-Length-Value* (TLV) structures**
- **Ethernet-Header (type 0x88cc), packets sent to multicast-address `01:80:c2:00:00:0e`**
- **Due to TLV stuff it was initially impossible to fuzz LLDP, with Spike and L2-addon**
- **=> addition of *s_binary_type_and_block_size_lldp()***
  - **gets an integer as the TLV-type**
  - **Plus char* as the name of the block**

LLDP (2)

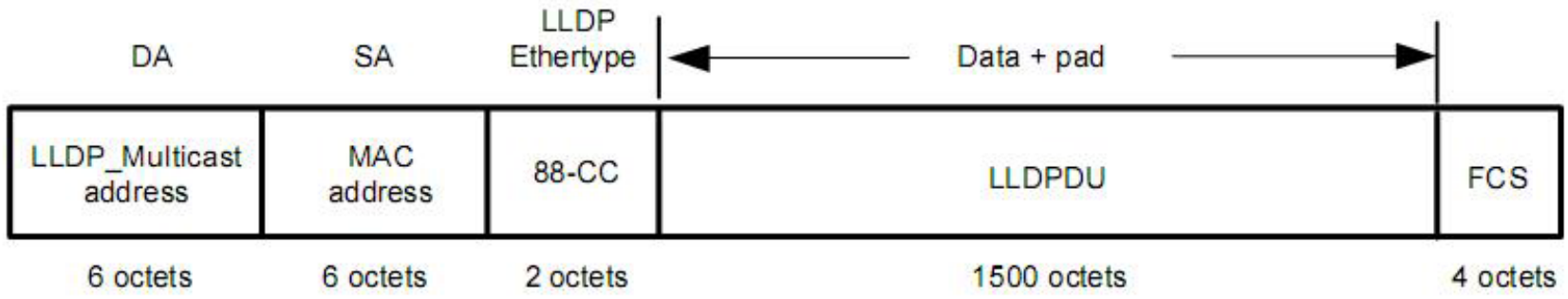
- **When multiple packets (containing different information) arrive from same source MAC address the packets are discarded**

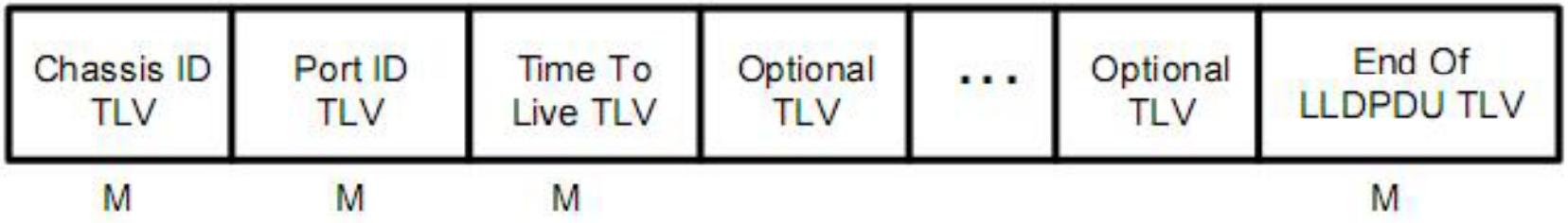
  => random source MACs needed
  => *generic_send_l2 rewritten* with *random_mac_option*

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

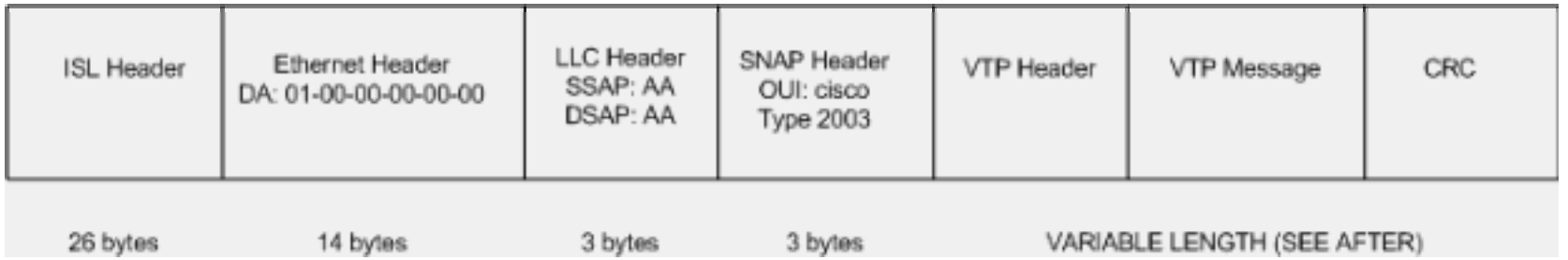
© ERNW GmbH . Breslauer Str. 28 . D-69124 Heidelberg . www.ernw.de

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
    */
```

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

ERNW Living Security.

Summary Advert Packet Format:

0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1			
Version	Code	Followers	MgmtD Len
Management Domain Name (zero-padded to 32 bytes)			
Configuration Revision Number			
Updater Identity			
Update Timestamp (12 bytes)			
MD5 Digest (16 bytes)			

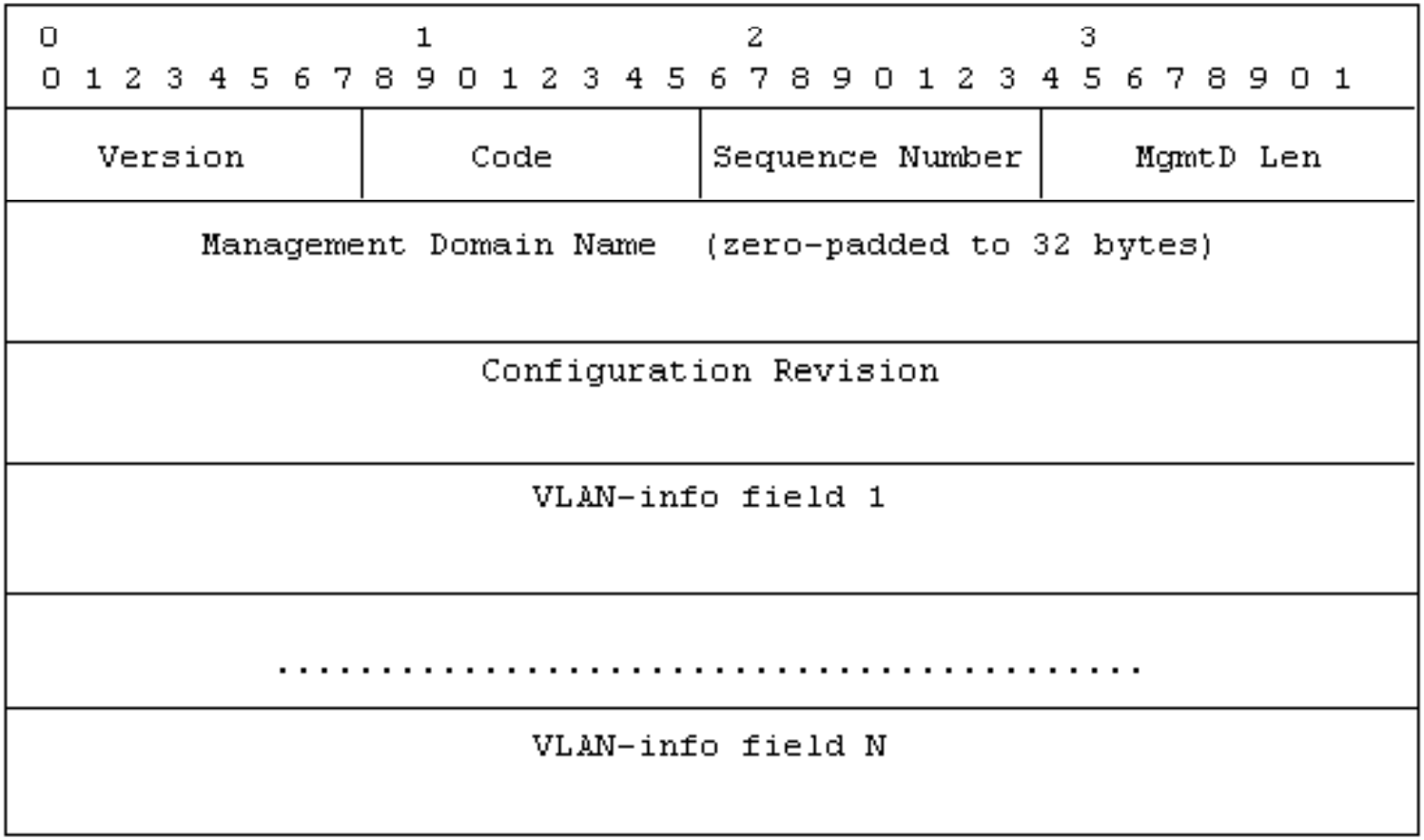
VTP packet format

- **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 revison 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("0000000000000000"); /* md5 digest / password - 16 bytes length */
```

Spike scripts
VTP Subset Request

ERNW Living Security.

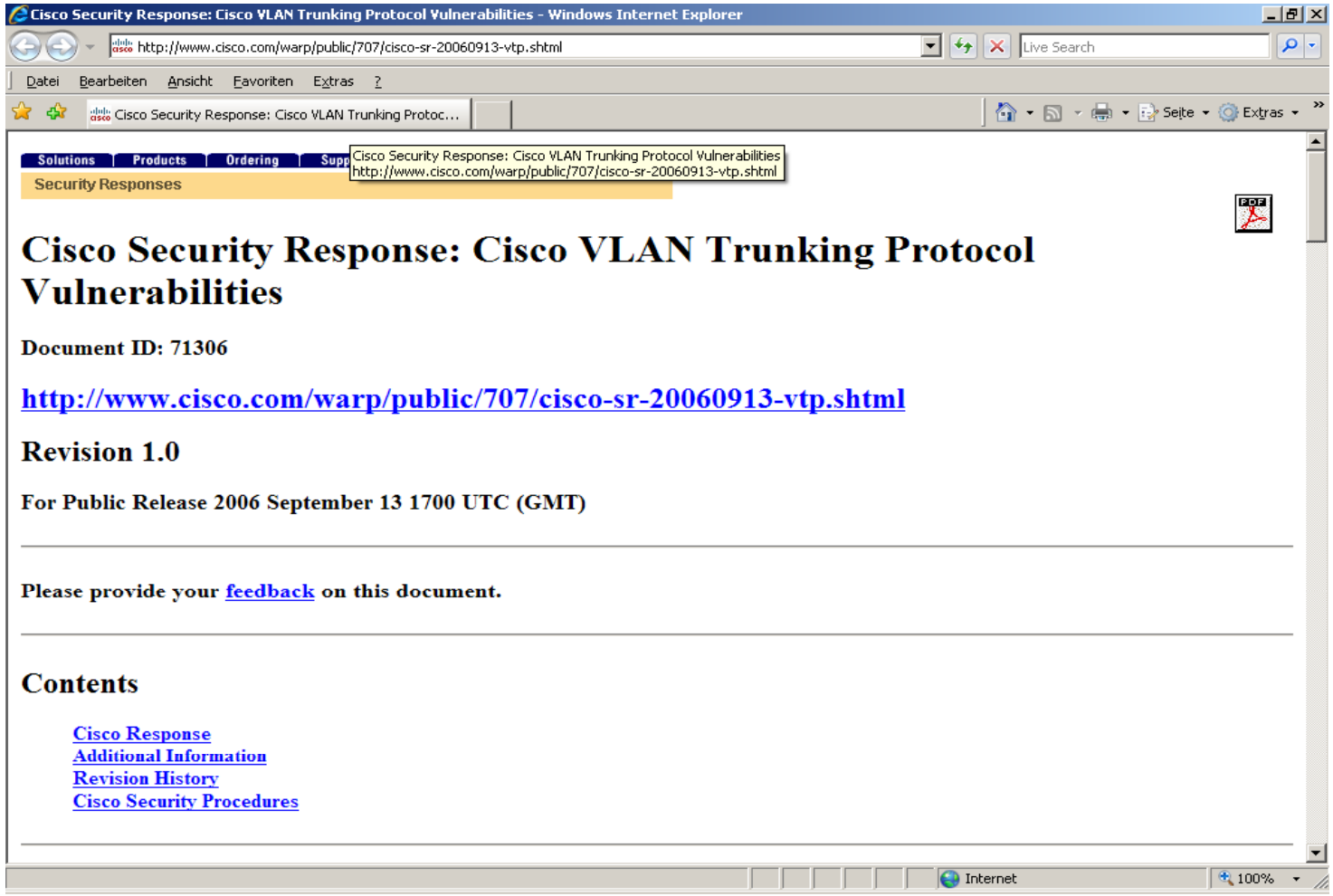
```
s_binary("aa"); /* DSAP */
s binary("aa"); /* SSAP */
s_binary("03"); /* func */
s_binary("00000c"); /* Orga-code */
s_binary("2003"); /* VTP */

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 (see above)**
- **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“

Further steps

- **Optimize VTP scripts (?)**

- **Test other vendors‘ devices**
- **Better fuzz strings particularly for nw devices?**

- **More prot definitions (anybody here any cycles left? ;-)**

Lessons Learned

ERNW Living Security.

- **Good documentation / comments rule**

- **Decide early what kind of fuzzer you need / want**
- **for a particular task**
  - **stateful**
- **for wider use**
  - **Stateless**

- **Debug the target that gets fuzzed**

The Code

- **Will the code be available?**
- **Yes! Now!**
- **We will put it on our website, too. Given it‘s a stress testing tool, no problems to expect with §202c...**
- **And we think about following the *n.runs* example anyway.**
- **We will continue developing this stuff and will add new protocol definitions (there are so many interesting L2 protocols out there ;-)**

Credits

ERNW Living Security.

- **To Daniel Mende for research and coding help**
- **To Dave Aitel for Spike**

- **To Angus for inviting us to Day-Con !!**

Questions?

ERNW
Living Security.

Thanks for your attention!

ERNW
Living Security.