LLDP Fuzzing

BSides Iowa 2019

New Presentation, Who Dis?

- Security Researcher at Aruba Threat Labs - Aruba Networks’ internal Red Team
- Professionally focused on developing offensive capabilities against network devices.
- Member of both SecDSM and Des Moines ISSA Chapter.
- IoT / networking hardware hacking enthusiast extraordinaire.
- Not good at expense reports.
- Looks like Shrek.

What is this Presentation?

This presentation is about **LLDP** and ways to **fuzz** the protocol to search for vulnerabilities.

We'll cover:

- What is Fuzzing?
- What is LLDP?
- How do I fuzz LLDP?
- What kinds of bugs will I find?
- Why would I want to do this in the first place?

Some Definitions

LLDP - Link Layer Discovery Protocol.

Fuzzing - Security technique that supplies mutated malicious input into a software program to find input that causes the program to crash.

What is LLDP?

The Link Layer Discovery Protocol is an OSI Layer 2 protocol.

It is on the same “lateral” as **ARP**.

It is used for advertising system information, including capabilities, to other network hardware connected at the link layer.

It helps network hardware know who is connected to it and where the management interface for connected hardware is located.

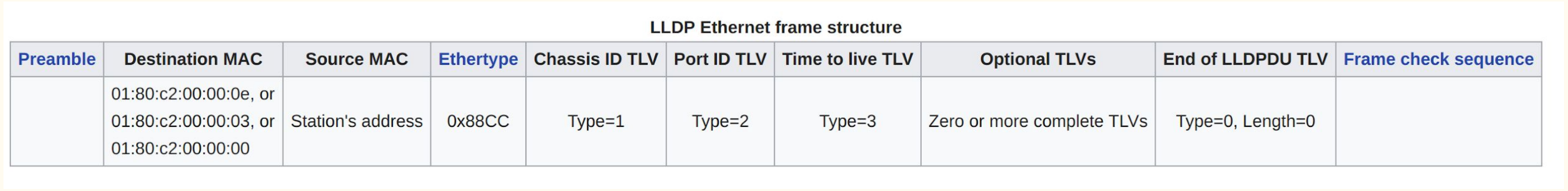
What is LLDP? (Cont)

LLDP can be considered a special type of Ethernet Frame.

It is important to note that LLDP does not work over WiFi, it is strictly a wired Ethernet connection protocol.

What is LLDP? (Cont)

LLDP Frame Structure:



LLDP Ethernet frame structure

Preamble	Destination MAC	Source MAC	Ethertype	Chassis ID TLV	Port ID TLV	Time to live TLV	Optional TLVs	End of LLDPDU TLV	Frame check sequence
	01:80:c2:00:00:0e, or 01:80:c2:00:00:03, or 01:80:c2:00:00:00	Station's address	0x88CC	Type=1	Type=2	Type=3	Zero or more complete TLVs	Type=0, Length=0	

Source: Wikipedia.org

How Does LLDP Work?

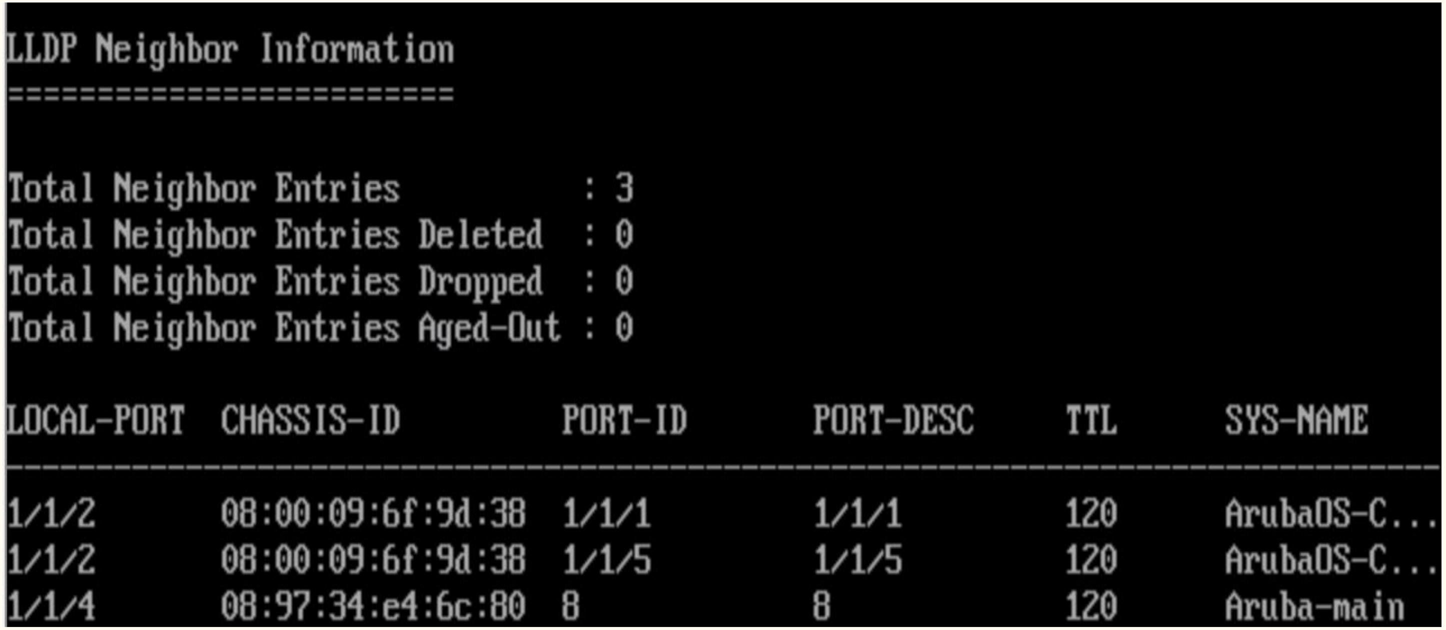
LLDP Specification lists three special values for MAC address that compliant switches know to look for for LLDP information.

The standard **IEEE 802.1AB** is not available for free.

https://standards.ieee.org/standard/802_1AB-2016.html

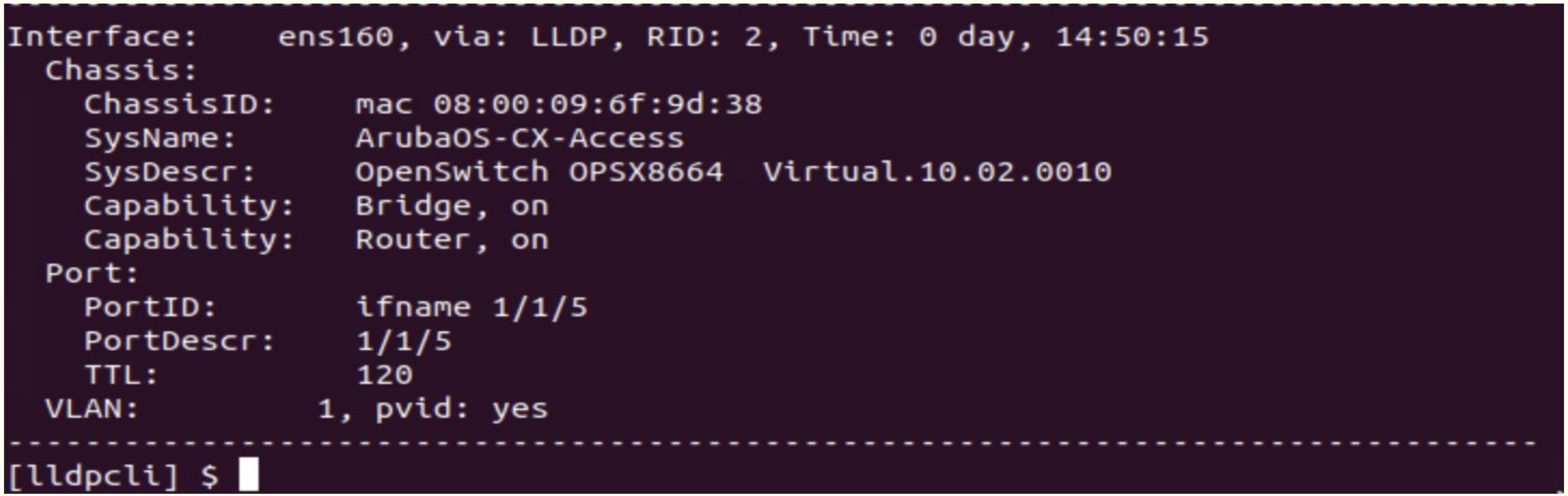
- **01:80:c2:00:00:0e**
- **01:80:c2:00:00:03**
- **01:80:c2:00:00:00**

What is LLDP? (Cont)



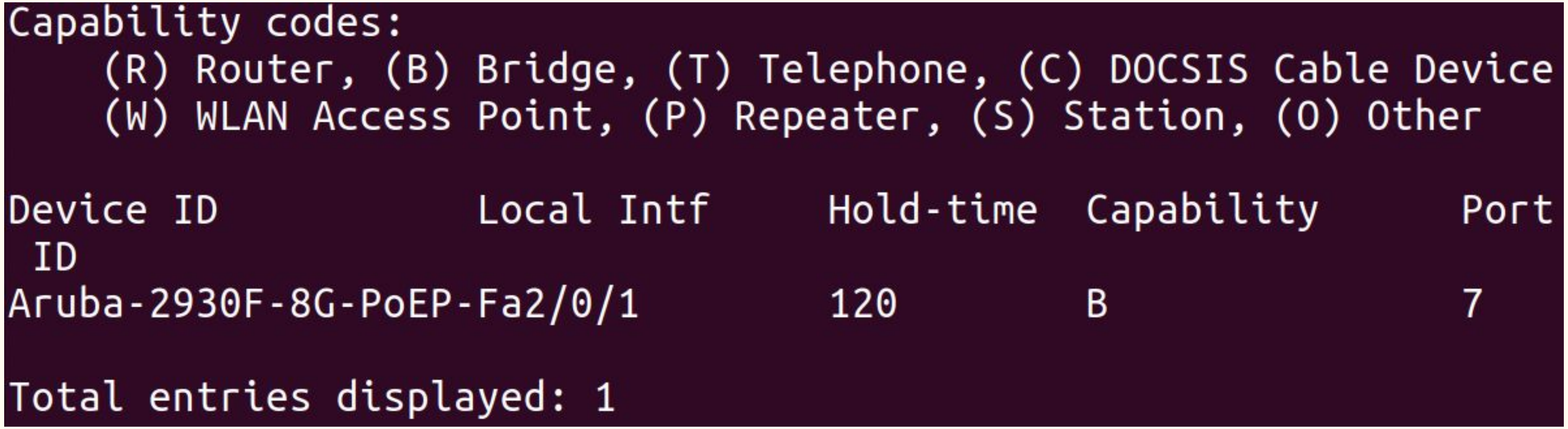
```
LLDP Neighbor Information
=========================

Total Neighbor Entries          : 3
Total Neighbor Entries Deleted  : 0
Total Neighbor Entries Dropped  : 0
Total Neighbor Entries Aged-Out : 0

LOCAL-PORT  CHASSIS-ID         PORT-ID        PORT-DESC      TTL      SYS-NAME
--------------------------------------------------------------------------------
1/1/2       08:00:09:6f:9d:38  1/1/1          1/1/1          120      ArubaOS-C...
1/1/2       08:00:09:6f:9d:38  1/1/5          1/1/5          120      ArubaOS-C...
1/1/4       08:97:34:e4:6c:80  8              8              120      Aruba-main
```

What is LLDP? (Cont)



```
Interface:    ens160, via: LLDP, RID: 2, Time: 0 day, 14:50:15
  Chassis:
    ChassisID:    mac 08:00:09:6f:9d:38
    SysName:      ArubaOS-CX-Access
    SysDescr:     OpenSwitch OPSX8664  Virtual.10.02.0010
    Capability:   Bridge, on
    Capability:   Router, on
  Port:
    PortID:       ifname 1/1/5
    PortDescr:    1/1/5
    TTL:          120
  VLAN:         1, pvid: yes
-------------------------------------------------------------------------------
[lldpcli] $
```

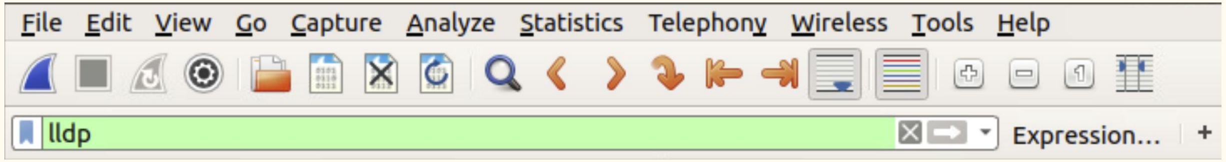
What is LLDP? (Cont)



```
Capability codes:
    (R) Router, (B) Bridge, (T) Telephone, (C) DOCSIS Cable Device
    (W) WLAN Access Point, (P) Repeater, (S) Station, (O) Other

Device ID           Local Intf     Hold-time  Capability      Port
 ID
Aruba-2930F-8G-PoEP-Fa2/0/1        120        B               7

Total entries displayed: 1
```

Wireshark Filter Expression

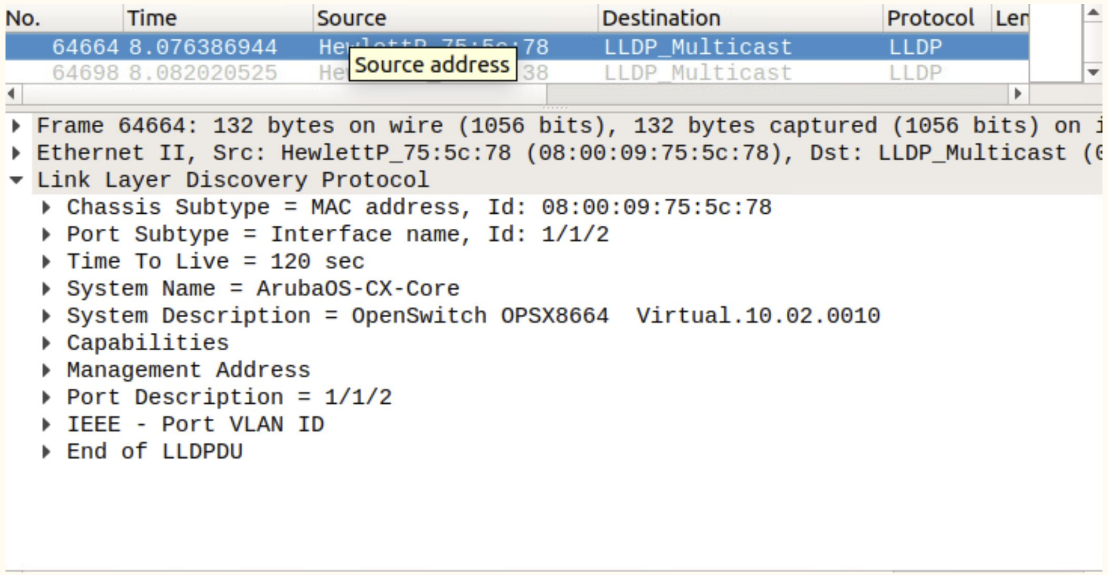
Use expression “**lldp**” to capture only LLDP traffic:



```
lldp
```

Wireshark Capture

This is what LLDP looks like in wireshark:



```
No.    Time         Source               Destination      Protocol  Ler
64664  8.076386944  HewlettP_75:5c:78    LLDP_Multicast   LLDP
64698  8.082020525  He                38 LLDP_Multicast   LLDP
                      Source address
```

```
▸ Frame 64664: 132 bytes on wire (1056 bits), 132 bytes captured (1056 bits) on i
▸ Ethernet II, Src: HewlettP_75:5c:78 (08:00:09:75:5c:78), Dst: LLDP_Multicast (0
▾ Link Layer Discovery Protocol
  ▸ Chassis Subtype = MAC address, Id: 08:00:09:75:5c:78
  ▸ Port Subtype = Interface name, Id: 1/1/2
  ▸ Time To Live = 120 sec
  ▸ System Name = ArubaOS-CX-Core
  ▸ System Description = OpenSwitch OPSX8664  Virtual.10.02.0010
  ▸ Capabilities
  ▸ Management Address
  ▸ Port Description = 1/1/2
  ▸ IEEE - Port VLAN ID
  ▸ End of LLDPDU
```

What is Fuzzing?

Fuzzing is the act of supplying randomly mutated input to a software program to see if it causes the program to crash.

The type of vulnerabilities found with this technique are almost always **memory corruption** vulnerabilities.

For more information on Fuzzing, see: https://fuzzing-project.org/

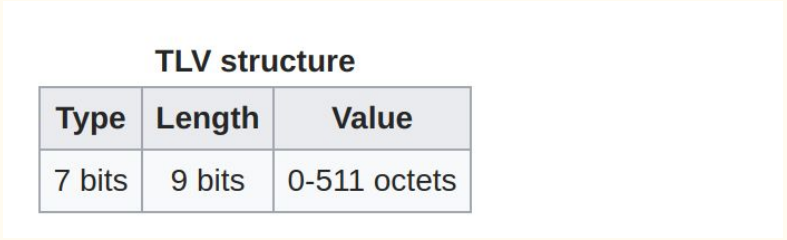
Great, what is LLDP used for?

From a practical standpoint, LLDP is typically used for debugging connectivity issues.

In large scale deployments, such as the data center, LLDP may drive some dynamic changes in network architecture using specialized software.

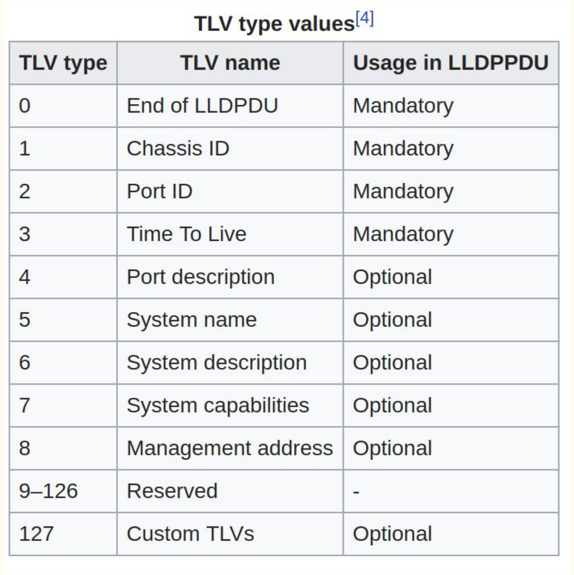
How do I fuzz LLDP?

The LLDP protocol has several **type-length-value** fields:



TLV structure

Type	Length	Value
7 bits	9 bits	0-511 octets



TLV type values[4]

TLV type	TLV name	Usage in LLDPPDU
0	End of LLDPDU	Mandatory
1	Chassis ID	Mandatory
2	Port ID	Mandatory
3	Time To Live	Mandatory
4	Port description	Optional
5	System name	Optional
6	System description	Optional
7	System capabilities	Optional
8	Management address	Optional
9–126	Reserved	-
127	Custom TLVs	Optional

Source: Wikipedia.org

How do I fuzz LLDP? (Cont)

So the **value** component of TLV fields can be 0-511 bytes long.

Also, the **length** component of TLV fields does not have to match the actual **value** component length.

How do I fuzz LLDP? (Cont)

Wouldn’t it be great if there was some sort of software that allowed us to craft arbitrary ethernet frames and send them over the wire?

Enter Scapy

https://scapy.net/

What is Scapy?

Pronounced “scay-pee” (thanks to Rick Farina for correcting me on this).

Scapy is a python library used to craft arbitrary network frames / packets and send them over the wire.

Scapy has built in support for LLDP.

- https://github.com/secdev/scapy
- https://github.com/secdev/scapy/blob/master/scapy/contrib/lldp.py
- https://github.com/secdev/scapy/blob/master/scapy/contrib/lldp.uts

Installing Scapy

```
$ pip install scapy
```

Or:

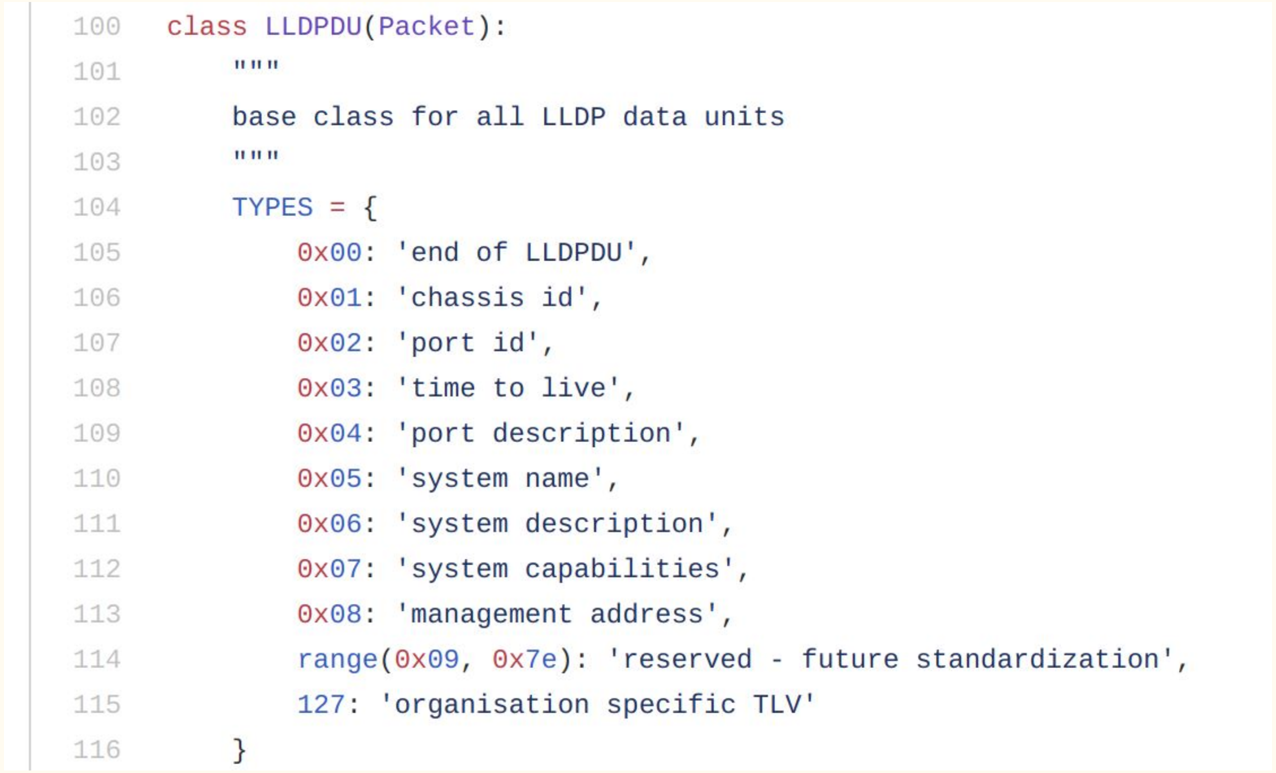
```
$ git clone https://github.com/secdev/scapy.git
```

```
$ cd scapy
```

```
$ sudo python setup.py install
```

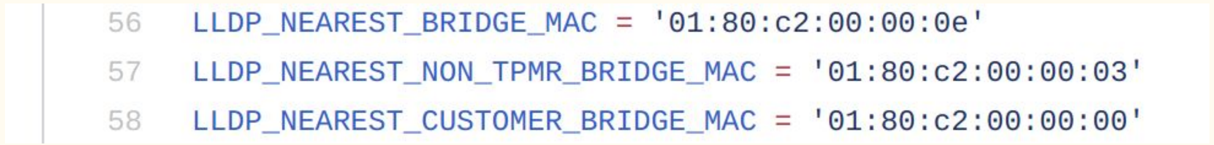
Note that the LLDP implementation was broken for one release around 2.4.0.

Scapy Source Example



```python
class LLDPDU(Packet):
    """
    base class for all LLDP data units
    """
    TYPES = {
        0x00: 'end of LLDPDU',
        0x01: 'chassis id',
        0x02: 'port id',
        0x03: 'time to live',
        0x04: 'port description',
        0x05: 'system name',
        0x06: 'system description',
        0x07: 'system capabilities',
        0x08: 'management address',
        range(0x09, 0x7e): 'reserved - future standardization',
        127: 'organisation specific TLV'
    }
```

Broadcast MAC Addresses



```
LLDP_NEAREST_BRIDGE_MAC = '01:80:c2:00:00:0e'
LLDP_NEAREST_NON_TPMR_BRIDGE_MAC = '01:80:c2:00:00:03'
LLDP_NEAREST_CUSTOMER_BRIDGE_MAC = '01:80:c2:00:00:00'
```

LLDP In Scapy

The LLDP implementation in Scapy has almost no documentation available on the internet for how to use it.

The LLDP test suite in Scapy (**lldp.uts**) provides some examples for how to use LLDP in scapy.

But to really understand the structure / syntax for LLDP frames, it is necessary to read the source (lldp.py).

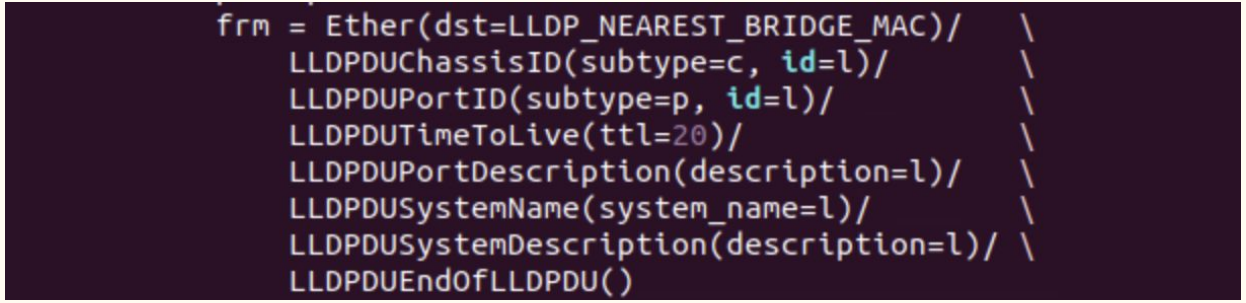
Scapy Source (Cont)

The LLDP module in Scapy is ~800 lines long.

Refer to it when you want to know arguments for a specific PDU.

LLDP in Scapy (Cont)

This is the basic syntax for crafting LLDP packets in Scapy:



```
frm = Ether(dst=LLDP_NEAREST_BRIDGE_MAC)/    \
    LLDPDUChassisID(subtype=c, id=l)/        \
    LLDPDUPortID(subtype=p, id=l)/           \
    LLDPDUTimeToLive(ttl=20)/                \
    LLDPDUPortDescription(description=l)/    \
    LLDPDUSystemName(system_name=l)/         \
    LLDPDUSystemDescription(description=l)/ \
    LLDPDUEndOfLLDPDU()
```

LLDP PDUs in Scapy:

- LLDPDUChassisID
- LLDPDUPortID
- LLDPDUTimeToLive
- LLDPDUSystemName
- LLDPDUSystemCapabilities
- LLDPDUManagementAddress
- LLDPDUEndOfLLDPDU

LLDP in Scapy (Cont)

So lets start fuzzing!

The first thing we want to check is the **length** component of the TLV. For this, we need to set:

```
conf.contribs['LLDP'].strict_mode_disable()
```

This enables us to create mismatching **length** parameters for the **value** length.

For example, we set **length** to be 1 and then the value is **`“A” * 254`**

Then, we do the copposite: set **length** to 254 and then the value to **`“A” * 1`**

LLDP Fuzzing in Scapy

The next thing we want to check is the **value** components for the various PDUs.

For example:

`LLDPDUSystemName` has the property **system_name**.

We can try and set this to common buffer overflow / string format values:

`“A” * 254`

`“%s%n” * 20`

Broadcast Addresses

Your fuzzer should take into consideration different logic flows for different values for the broadcast mac address. Remember, there are only three different valid values for LLDP traffic.

TLV fuzzing

You will want to fuzz each part of a **TLV**:

- Type: Integer between 0-127
- Length: Integer between 0-511
- Value: any string $<$ 511 bytes long

Other PDUs have different parameters

You will need to review the **LLDP.py** file to learn all the parameters for the various PDUs.

Each parameter is typed, so you will get a interpreter warning at runtime if you set a parameter to an invalid value.

This is where automation via a python script can come in handy.

Run LLDP Fuzzer, Run!

After you've constructed your python script, you will need to run it on a machine that has an ethernet link to another machine (can be virtual).

What you should watch for is the serial console output from the switch / machine you are sending the LLDP data to.

LLDP data will not affect the host that the LLDP originates from.

Crash!

While monitoring the Serial Console output, it will be obvious when you have a crash.

You will see all kinds of strange stack and register information, and a coredump is usually generated and stored on the switch.

Many switches can be configured to send coredumps off to a **FTP** or **TFTP** server after recovery.

Finding the “guilty” input

Since there is really no way to comprehensively and definitively determine when a crash has occurred other than by manual serial console observation, it may be difficult to determine which LLDP input caused the crash.

I recommend sleeping the thread between LLDP frames:

```
time.sleep(10)
```

Show commands

In some LLDP implementations, just sending bad data to over the wire will cause a crash.

In other implementations, it is necessary to run something like:

```
# show lldp neighbors
```

On the switch to cause a crash. Both vectors should be evaluated.

Bugs

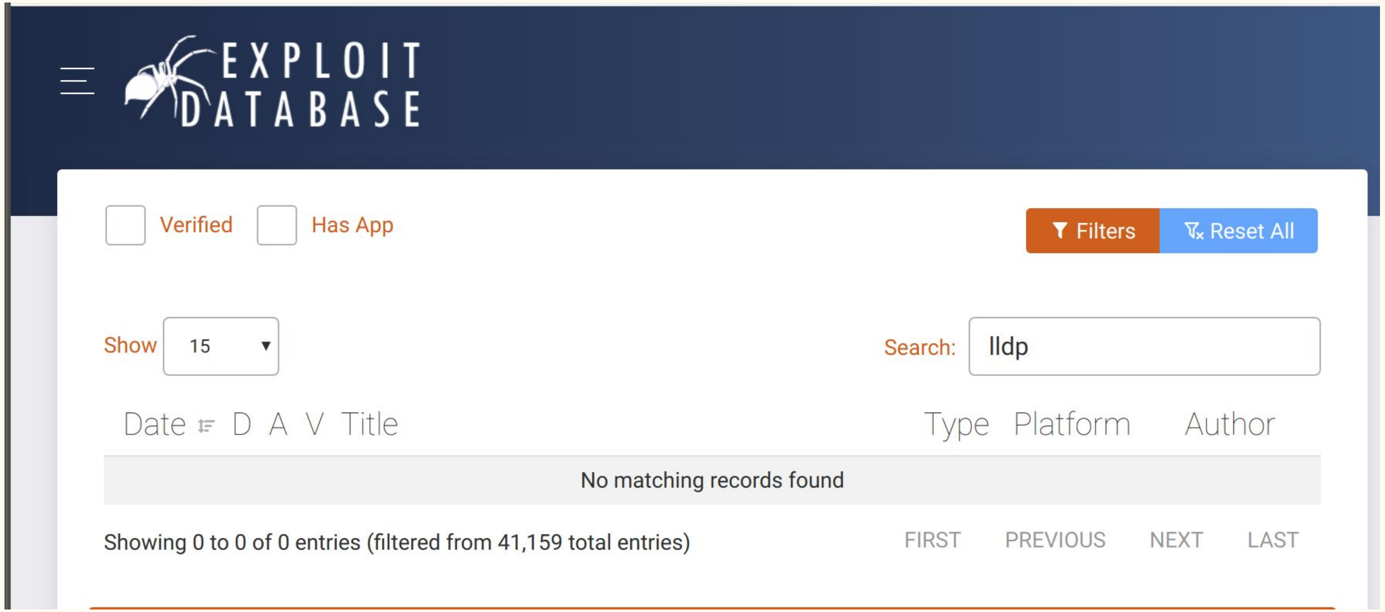
As mentioned before, the typical types of bugs that come out of fuzzing LLDP are memory corruption bugs.

This include typical stack buffer overflows as well as heap buffer overflows.

Most switches run a proprietary real time operating system, which makes exploit development difficult, as an exploit written for one device on one firmware version may not work on the next firmware version, let alone another device model.

Bugs (Cont)

There are no published exploits that work over LLDP:



Date	D	A	V	Title	Type	Platform	Author
No matching records found							

Showing 0 to 0 of 0 entries (filtered from 41,159 total entries)

Blast Radius

Attacking a switch or server over LLDP requires an adjacent position on the network, connected by ethernet. As such, LLDP attacks will not work over the public internet, as LLDP is a LAN protocol.

This means you may need a separate foothold on the target network to launch exploits over LLDP.

Why would anyone want to do this?

The primary goal of launching an exploit over LLDP would be to compromise network infrastructure, as a general rule. While LLDP implementations exist for servers, they are most commonly found on switches and routers.

Taking over a switch or a router gives an attacker an extremely privileged view on the network. Not only can they view traffic traversing the network device, but in some cases they can modify it, or drop the packets to deny service.

Takeaways

- Attacks can happen at layer 2.
- Would your network monitoring solution have caught this type of attack?
- Fuzzing LLDP implementations on vendor hardware will bear more fruit than attacking open source implementations.
- It would be really, really difficult to automate fuzzing completely.
- Scapy is a powerful tool that can craft all kinds of frames / packets.

Questions?

Thank you for attending my presentation!

https://twitter.com/nstarke

https://github.com/nstarke