The University of Texas at Austin, Department of Computer Sciences, Technical Report TR-07-24

A Link Layer Discovery Protocol Fuzzer

Jeremy Hollander
Department of Computer Sciences
The University of Texas at Austin

***Abstract*— We developed the first Link Layer Discovery Protocol (LLDP) fuzzer with ten test cases to find security vulnerabilities in LLDP-enabled network devices. The current test cases look for off-by-one errors, consistency errors, buffer overflows and stack injections. Furthermore, our fuzzer can easily be extended with additional test cases.**

***Index terms*— Link Layer Discovery Protocol, Fuzzer, Security, Penetration Testing**

I. Introduction

A common issue for system and network administrators is finding out which devices are connected to their network and finding out more information about these. It would be especially useful for network administrators in large organizations with thousands of connected nodes to discover which devices are connected as well as data such as their MAC address and IP address.

In 1994 Cisco Systems introduced in their network equipment a new layer two protocol which they call the Cisco Discovery Protocol (CDP) [1]. The purpose of this protocol is to share information about the equipment with other devices by multicasting messages such that each device can learn about other devices on the local network. Other network manufacturers such as Foundry Networks and Nortel Networks followed suit with FDP [2] and NDP respectively. While each of these protocols use a similar structure they contain significant differences such as the MAC address messages are sent to, the type of information that is sent out and how information is accessed. This is one of the reasons the IEEE ratified the Link Layer Discovery Protocol (LLDP) as 802.1AB in 2005 [3].

The Link Layer Discovery Protocol incorporates all of the advantages of the three protocols mentioned above in addition to being standardized such that devices from different manufacturers are able to exchange information with one another. One major problem with LLDP however is that messages which devices send out are not authenticated. This makes it easy for malicious users to send out LLDP packets purporting to be from some device on the local network. Some administrators may perform tasks based on the information they received using LLDP. This may in some cases be fatal for the network device whose identity was forged in an LLDP packet. For instance, a malicious user may choose to forge a message in which it states that some router supports Power over Ethernet. The administrator may then decide to transmit electrical power over the twisted pair cable to that device in order to switch it on. If the router does not support that functionality this may prove to be lethal for that device. At this time there is however no plan to introduce an authenticated version of LLDP. An additional security issue is that recipients of the LLDP packets may not always check that the information obtained adheres to the protocol prior to storing it locally. This may lead to a large number of security vulnerabilities as the malicious user may inject any type of information in any LLDP recipient. Both of these problems lead to the development of our Link Layer Discovery Protocol fuzzer which we present in this paper.

In the next section we provide a summary of LLDP. In the third section we briefly introduce the reader to fuzzers following which we explain the structure and test cases included in our fuzzer. In the fourth section we show how to simply and efficiently add test cases. Finally, we conclude on our work and provide some ideas for future work in the fifth section.

II. The Link Layer Discovery Protocol

The Link Layer Discovery Protocol is one-way such that an agent is not able to solicit information from other LLDP agents. An agent only ever transmits information about the current status of the device it is active on. Information which is transmitted and received is stored in a Management Information Base (MIB) to allow it to be accessed by a Network Management System (NMS) using a protocol such as the Simple Network Management Protocol (SNMP). While there are several MIBs in an LLDP-enabled device only two are required for the protocol to function correctly: the LLDP local system MIB and the LLDP remote system MIB. The former stores information about the local device, which is transmitted on the local network, while the latter stores information received from neighboring devices.

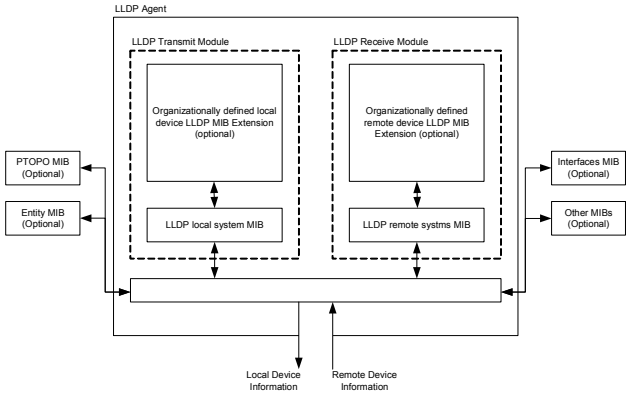

Fig. 1 — The LLDP Agent

An LLDP packet contains two major types of information: the LLDP Ethertype (0x88cc) and the LLDP Data Unit (LLDPDU). An LLDP packet is always sent out to the same Multicast address to ensure that it is not forwarded by MAC Bridges that conform to the IEEE 802.1D-2004 standard. The LLDPDU consists of a series of information elements known as TLVs, which stands for Type-Length-Value. Each element contains the following three fields:

1. The type of data being sent
2. The length in bytes of the data being sent
3. The value of the information that is being sent

Each LLDPDU must contain four compulsory TLVs in addition to any number of optional TLVs. The four compulsory TLVs are the Chassis ID TLV, the Port ID TLV, the Time To Live TLV and the End of LLDPDU TLV. They must always appear in that specific order.

The first compulsory TLV, the Chassis ID, whose TLV type is 1, identifies the chassis containing the LLDP agent. There are several ways to identify the chassis, one of which is by its MAC address. Other types of information that may be provided include the Interface alias, Port Component and Network Address. The TLV information string's length may range between 1 and 255 bytes.

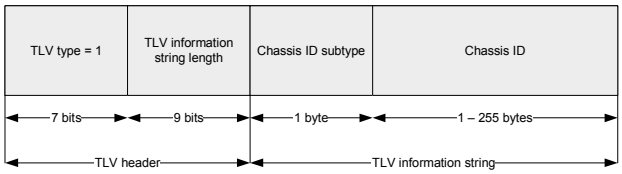

Fig. 2 — The Chassis ID TLV

The second compulsory TLV, the Port ID, whose TLV type is 2, identifies the port component of the MAC Service Access Point associated with the transmitting LLDP agent. This TLV is very similar to the Chassis ID with the difference being that it works at a more refined level. In addition the transmitting agent is able to send an Agent Circuit ID. RFC 3046 [4] specifies that it is used to encode an agent-local identifier of the circuit from which a DHCP client-to-server packet was received. It is intended for use by agents in relaying DHCP responses back to the proper circuit. The size of the information string length may also vary between 1 and 255 bytes.

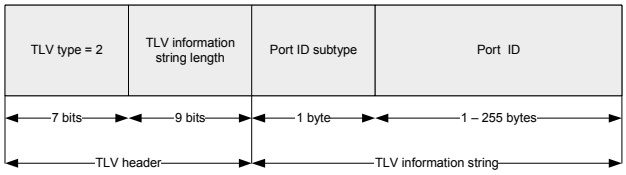

Fig. 3 — The Port ID TLV

The third compulsory TLV, the Time To Live (TTL) TLV, whose TLV type is 3, specifies in seconds how long the information contained in an LLDP packet may be considered fresh. For instance, if the TTL is 60 then the recipient agent may automatically discard the information received after 60 seconds as it is considered out-of-date. If an LLDP packet with a TTL of 0 is received the agent may delete all the information associated with the sender.

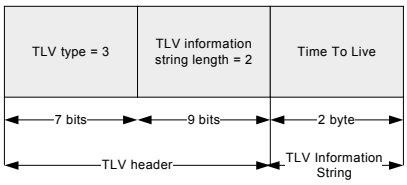

Fig. 4 — The TTL TLV

The fourth and final compulsory TLV, the End of LLDPDU TLV, whose TLV type is 0, is used to mark the end of the LLDP packet. Because there is never any payload attached to this TLV the information string length is always 0.

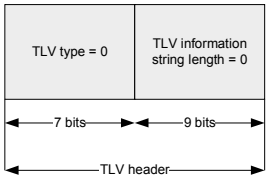

Fig. 5 — The End of LLDPDU TLV

As shown in figure 6 a number of optional TLVs may be added to the LLDP packet between the TTL TLV and the End of LLDPDU. These optional TLVs are the Port Description TLV, System Name TLV, System Description TLV, System Capabilities TLV, Management Address TLV and Organizationally Specific TLVs.

Fig. 6 — The compulsory and optional TLVs

III. THE LLDP FUZZER

A. Introduction to fuzzing

Fuzzing is a black-box testing technique used to find bugs in software or hardware devices. While regular testing techniques must still be used to find bugs, fuzzing is used to discover input combinations which the developers of the software or hardware device under test may not have taken into consideration. Prior to building a fuzzer one must understand the protocol used to attack the device under test. Once knowledge of the protocol has been acquired the fuzzer's architecture can be built. Using the fuzzer security researchers generate well-formed and malformed network packets aimed at the device under test in order to find out whether it was properly set up to handle input of legal as well as illegal data.

B. The LLDP Fuzzer

The LLDP fuzzer we developed aims to find security vulnerabilities in the LLDP receiving agent by sending malformed packets. We have devised a specific set of ten test cases which we feel would find vulnerabilities in the device under test. Any one of these test cases could prove fatal for the receiving agent if illegal or malicious data is not handled properly or not discovered prior to its insertion in the MIB. In addition, we shall show in the next section how to add new test cases.

1) Test Case 1

In the first test case we overload the payload for the Chassis ID TLV. The LLDP specification specifies that the maximum payload size for the Chassis ID TLV is 255 bytes. In this first test case we send 510 bytes. We correctly store the information string length as being 511 bytes. If the receiving agent does not perform any verification on the size of the Chassis ID TLV when receiving LLDP packets it may only assign 255 bytes for this TLV. This may have as a result that the Port ID TLV, or any information stored in sequence after the Chassis ID TLV, will be overwritten with the last 255 bytes of the information string.

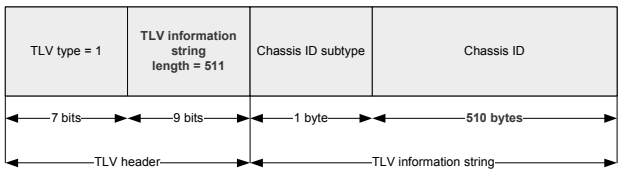

Fig. 7 — Test case 1, sending an overloaded payload for the Chassis ID TLV

2) Test Case 2

In the second test case we send an LLDP packet which contains a Chassis ID TLV with no payload. On parsing this TLV we should expect the receiving agent to discard the message because the minimum TLV information string length is two bytes: one byte for the Chassis ID subtype and at least one byte for the information string.

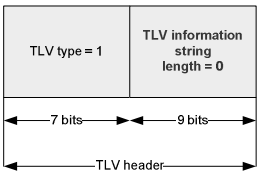

Fig. 8 — Test case 2, sending a Chassis ID TLV with no payload

3) Test Case 3

In the third test case we investigate whether the TLV information string length is tested against the real size of the information string in the Chassis ID. In this packet we set the TLV information string length to 1 such that the receiving agent may only reserve a single byte for this TLV. However we provide a four-byte payload. Similarly to the first test case it is possible that the Chassis ID information string will overwrite the Port ID TLV, if both are stored in sequence in the MIB. Otherwise it may overwrite random information in the MIB.

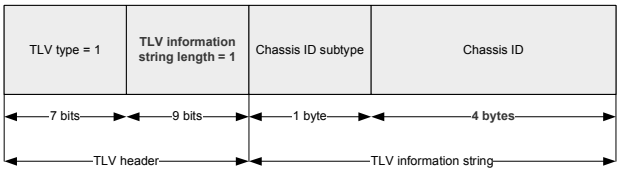

Fig. 9 — Test case 3, sending a false information string length

4) Test Case 4

In the fourth test case we send a malformed packet with a 256 byte Chassis ID information string. This is one byte larger than allowed by the protocol. The information string length is therefore, including the Chassis ID subtype, 257 bytes. In this case we wish to find out if the device under test may have an off-by-one error.

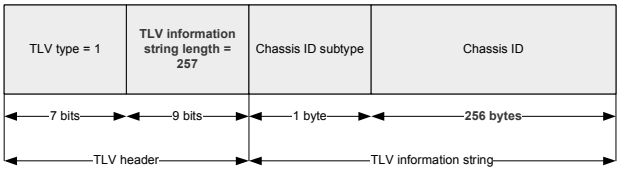

Fig. 10 — Test case 4, sending an overloaded payload

5) Test Case 5

In the fifth test case we send a burst of 1000 LLDP packets with a TTL TLV of 15 seconds. While each LLDP packet is legal according to the protocol we wish to test whether the receiving agent may have a mechanism in place which would restrict it from receiving a large amount of packets in a short period of time, especially if the Time To Live TLV states that the information received is fresh for 15 seconds.

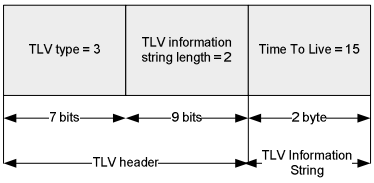

Fig. 11 — Test case 5, sending a burst of packets with a TTL of 15 seconds

6) Test Case 6

According to the LLDP specification the End Of LLDPDU TLV may never contain any payload. In the sixth test case we send a packet with a two-byte payload attached to the End Of LLDPDU TLV. In addition we leave the information string length field to zero. Since the End Of LLDPDU TLV is always placed at the end of a packet it has a crucial location in terms of finding a vulnerability in the remote system. Because the recipient of the packet may not allocate any storage for the payload as the information string length states that there is no payload, it is possible that the payload which in this case is two bytes long may overwrite some data located on the recipient's executable stack. This may prove to be fatal as the recipient may unknowingly execute malicious code injected by the sender in the payload of this packet if that payload overwrites memory from the executable stack.

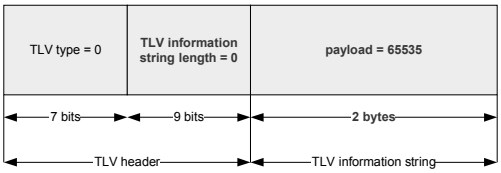

Fig. 12 — Test case 6, sending an End of LLDPDU with a non-zero payload and zero information string length

7) Test Case 7

The seventh test case is very similar in fashion to the previous test case. The only difference is that we provide the actual payload in the information string length. The purpose of this test case is to determine whether the recipient of this LLDP packet would malfunction if it receives an End Of LLDPDU TLV with an information string length not equal to zero.

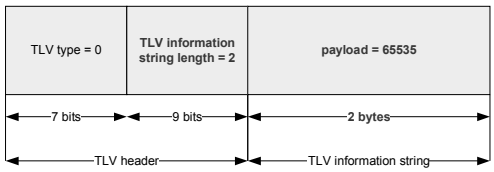

Fig. 13 — Test case 7, sending an End of LLDPDU with payload and information string length equal to 2

8) Test Case 8

In the eighth test case we send an illegal System Capabilities TLV information string. The optional System Capabilities TLV is used to identify the primary functions of the sender and whether or not these primary functions are enabled. These functions may include a repeater capability, bridge capability, wireless LAN access point capability or router capability. There are in total seven functions with another eight reserved for future use. An example of a legal message would include information about a device being capable of acting as a bridge and wireless LAN access point however at the time the message is sent only the wireless LAN access point functionality is enabled. In this test case the malformed packet specifies that the sender's system may only function as a bridge (the third lowest bit specifies the bridge functionality) however at the time the message is sent the bridge and wireless LAN access point functionalities enabled (the fourth lowest bit specifies the wireless LAN access point functionality). This is an inconsistency which must be rejected by the recipient of this packet. In this test case we attempt to determine whether the recipient has consistency-checks in place. If no such checks are present an error may occur.

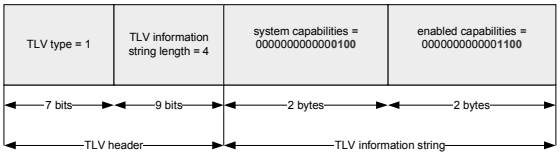

Fig. 14 — Test case 8, sending a malformed System Capabilities TLV

9) Test Case 9

In the ninth test case we wish to find out whether the recipient's LLDP agent has implemented the protocol at its most basic level. In this test case the Port ID TLV is missing. According to the specification the LLDPDU must be checked to ensure that it contains the correct sequence of mandatory TLVs.

Fig. 15 — Test case 9, sending an LLDPDU with the Port ID TLV missing

10) Test Case 10

The tenth test case sends a Chassis ID TLV with an IPv4 address yet specifies that it is providing an IPv6 address. The purpose of this test case is to find out whether the recipient's LLDP agent will fail upon receiving the wrong type of IP address even though the address provided is a valid version 4 address.

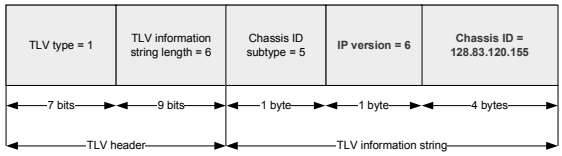

Fig. 16 — Test case 10, sending a Chassis ID TLV with a false IP version

IV. How To Extend the LLDP Fuzzer

While we have written ten test cases for the LLDP fuzzer a major part of this work is also to provide the necessary architecture to allow security researchers to complement the existing test cases with their own. We have ensured that doing so is straightforward by automatically setting default values for all the compulsory and optional TLVs when instantiating a new LLDP packet. The advantage is two-fold. First, it is not necessary to set each TLV each time one creates a new test case. Second, when performing security testing one will usually only modify a single parameter in a test case such that every single TLV but the one under test will remain similar across several test cases. This is done in order to easily pinpoint to the source of a vulnerability, if one is found. The following test case demonstrates the steps required to write a new test case in the LLDP fuzzer.

```
1. def test_case7():
2.     p=lldppacket()
3.     p.end_of_lldpdu_data_customsize=2
4.     p.mod_end_of_lldpdu()
5.     p.mod_assemble_packet()
6.     p.packet+=new_field(0,65535,"short")
7.     p.send_packet(self_iface,self_mac)
```

In the first line we define a new test case. In the second line we instantiate a new LLDP packet. This new packet comes along with default values set for each compulsory and optional TLV. For instance, the default value for the Time To Live TLV is 60 seconds. In the third line we set the information string length of the End Of LLDPDU TLV to 2 bytes. The mod_end_of_lldpdu() function in the fourth line is used to store the modifications in the End Of LLDPDU TLV. Similarly, if one were to change the value of the Time To Live TLV from 60 seconds to 120 seconds one would first set p.ttl_payload_data to 120 followed by p.mod_ttl(). Because we have modified a TLV in the LLDP packet we must regenerate the packet so we use mod_assemble_packet() once we have performed all the necessary changes in the TLVs. This function assembles the LLDP packet by running through all the TLVs. This test case is somewhat unusual as we add a payload to the End Of LLDPDU. Thus, in the sixth line, following the assembly of the packet, we add a two-byte field with the largest integer value that can fit in such a field. Finally, in the seventh line we send the packet. Two parameters must always be provided when sending an LLDP packet: the interface on which to send the packet and the MAC address from which it should appear to be sent from.

In addition to being able to add new test cases one may also create a wrapper to run through a list of input combinations. For example, one may wish to send LLDP packets with a combination of legal and optional TLVs as well as TLVs which are currently unassigned. The type numbers for the reserved (unassigned) TLVs range from 9 to 126.

V. Conclusion and Future Work

We have shown our LLDP fuzzer and ten test cases we devised for it. While we have been unable to test our fuzzer against a network device we strongly believe that at least one of our test cases will prove to be problematic when tested on an LLDP-enabled device. We have furthermore also created an easy-to-use architecture such that security researchers may add their own test cases as more LLDP-compliant devices arise on the market.

The purpose of the LLDP fuzzer is to find security vulnerabilities which may not be easily found using regular testing techniques. The test cases we have devised attempt to find out whether the device under test is prone to off-by-one errors, consistency errors, buffer overflows and stack injections.

While we have provided a fuzzing architecture for LLDP, as there is currently no way to tell whether the recipients of the LLDP packets have crashed or misbehaved it would be useful to run a module on receiving agents. This module could send back the information that was stored in the LLDP local MIB such that it may be easier to analyze the results of the vulnerability testing. In addition, as more LLDP-enabled devices emerge, newer test cases should be devised.

VI. References

[1] Cisco Systems, “The Cisco Discovery Protocol Packet Format”, http://www.cisco.com/univercd/cc/td/doc/product/lan/trsrb/frames.htm#xtocid12, viewed on 04/30/2007

[2] Foundry Networks, “Enabling the Foundry Discovery Protocol”, http://www.foundrynet.com/services/documentation/bigiron_rx_config/CDP_FDP.html, viewed on 04/30/2007

[3] The IEEE, “The Link Layer Discovery Protocol”, http://standards.ieee.org/getieee802/download/802.1AB-2005.pdf, viewed on 04/30/2007

[4] M. Patrick, “DHCP Relay Agent Information Option”, RFC 3046, January 2001