ERNW
providing security.

Friedwa
rt Kuhn

Digital unterschrieben von Friedwart Kuhn
DN: c=DE, o=ERNW Enno Rey Netzwerke GmbH, cn=Friedwart Kuhn, givenName=Friedwart Nikodemus Michael, sn=Kuhn, serialNumber=DTRWM384176041574922
Datum: 2015.03.06 14:12:00 +01'00'

# ERNW Newsletter 48 / March 2015

## An MLD Testing Methodology

| | |
|---|---|
| Date: | 3/6/2015 |
| Classification: | Public |
| Author(s): | Dr Antonios Atlasis & Jayson Salazar |

ERNW
providing security.

# Table of Content

## List of Figures

ERNW
providing security.

## List of Tables

# 1 INTRODUCTION

Multicasting is one of the key functionalities in IPv6. As explained in IPv6 specification, RFC 2460, the scalability of multicast routing is improved by adding a "scope" field to multicast addresses". IPv6 multicast addresses and specifically the so called solicited-node multicast addresses are used by one of the most important IPv6 functionalities, the Neighbor Discovery process (RFC 4861).

*Multicast Listener Discovery* (MLD) is one of the sub-protocols of the IPv6 family. As it is defined in the corresponding specifications (RFC 2710, RFC 3810), MLD is used by IPv6 routers to discover the presence of multicast listeners and specifically, which multicast addresses are of interest to the neighboring nodes of the MLD capable routers. This information is then provided to whichever multicast routing protocol (like PIM) is being used by the router for other (e.g. WAN) communication.

Each router keeps a list for each attached link of which multicast addresses there are listeners on that link. A router only needs to know that listeners for a given multicast address are present on a link, and not the identity (e.g. unicast addresses) of these listeners, or not even how many listeners are present.

MLD was first defined in RFC 2710 and it was derived from IPv4's IGMPv2. However, MLD uses ICMPv6 (IP Protocol 58) message types as an underlying protocol. Later, in RFC 3810 MLDv2 was defined.

MLDv2 is designed to be interoperable with MLDv1. MLDv2 adds the ability for a source filtering, as required to support Source-Specific Multicast (RFC 3569). This allows a node to report interest in listening to packets with a particular multicast address only from specific source addresses or from all sources except for specific source addresses.

MLDv2 is a translation of the IGMPv3 protocol (RFC 3376) for IPv6 semantics.

For the rest of this document, when we want to refer to the initial specification (RFC 2710) or the second version of MLD (RFC 3810) we will use the terms MLDv1 and MLDv2 respectively. On the contrary. when using the term MLD, this will refer to both versions.

There is a lot of discussion regarding the real necessity of MLD for core IPv6 functionalities, like the Neighbor Discovery process. It is beyond the scope of this document to try to discuss this issue further. However, it is a fact that in most modern Operating Systems (OS) – with the exception of OpenBSD, MLD is enabled by default. Moreover, in most of the cases MLD cannot even be disabled without breaking the Neighbor Discovery process. Furthermore, as defined in RFC 3810, MLD messages are not subject to source filtering and must always be processed by hosts and routers. So, even if an IPv6 environment does not use typical multicast applications (like IPTV, video-conference, etc.), MLD is there and hence, all the used MLD-capable devices, including routers, OS, or even switches (when MLD snooping is enabled) should be carefully evaluated regarding their MLD operation.

The goal of this document is to provide a concise methodology on testing MLD capable devices from a security perspective. Although a lot of different testing cases are described both for MLDv1 and MLDv2, by no means can the list of the presented scenarios be considered as an exhaustive one. However, it can serve as a really good basis for testing your MLD devices and, if required, expanding the tests.

# 2 MLD FUNDAMENTALS

It is beyond the scope of this document to give a full description of the MLD protocol. However, some basic informational background will be given which will allow the reader to follow smoothly the description of the MLD testing methodology. For further information, the reader is advised to go through RFC 2710 and RFC 3810 for MLDv1 and MLDv2 respectively.

MLD is an asymmetric protocol (different behaviors for multicast listeners and routers).

As already mentioned, MLD uses ICMPv6 as an underlying protocol.

In the rest of this section, for reasons of completeness of this document a brief description of the MLD messages will be given, while the role of Queriers, which are critical for MLD's operation, will also be explained.

## 2.1 Types of MLD Messages

### 2.1.1 MLDv1 Types of Messages

In MLDv1 three types of messages are defined:

- *Multicast Listener Query* (Type 130): They are used to query listeners for multicast addresses those are interested in. There are two sub-types:
  - *General Query*, used to learn which multicast addresses have listeners on an attached link. It is sent to the ff02::1 multicast address (link-scope all-nodes).
  - *Multicast-Address-Specific Query*, used to learn if a particular multicast address has any listeners on an attached link. It is sent to the specific multicast address being queried.
- *Multicast Listener Report* (Type 131): It is used by listeners to report multicast addresses they are interested in. These are sent to multicast address being reported.
- *Multicast Listener Done* (Type 132): They are used by listeners to report that they are no longer interested in listening to a specific multicast address. These are sent to FF02::2 multicast address (link-scope all-routers).

The length of MLDv1 messages must not exceed 24 octets.

The format of the MLDv1 messages is given in Appendix B.

### 2.1.2 MLDv2 Types of Messages

There are two MLDv2 message types:

- *Multicast Listener Query* (Type 130): They are used to query listeners for multicast addresses those are interested in. There are three types of MLDv2 query messages: General Queries, Multicast Address Specific Queries, and Multicast Address and Source Specific Queries.
  - *General Queries* are sent to the link-scope all-nodes multicast address: FF02::1.
  - A *Multicast Address Specific Query* is used to verify whether there are nodes still listening to a specified multicast address or not.
  - A *Multicast Address and Source Specific Query* is used to verify whether, for a specified multicast address, there are nodes still listening to a specific set of sources, or not.
- Version 2 *Multicast Listener Report* (Type 143). MLDv2 Reports are sent with an IP destination address of FF02::16, to which all MLDv2-capable multicast routers listen.

The format of the MLDv2 messages is given in Appendix B.

As the reader can notice, there are no MLD Done messages in MLDv2. Instead, each node maintains or computes multicast listening state for each of its interfaces. This state consists of a set of records, with each record containing an IPv6 multicast address, a filter mode, and a source list. The filter mode may be either INCLUDE or EXCLUDE. In INCLUDE mode, reception of packets sent to the specified multicast address is enabled only from the source addresses listed in the source list. In EXCLUDE mode, reception of packets sent to the given multicast address is enabled from all source addresses except from those listed in the source list. So, the equivalent of MLDv1 Done messages in MLDv2 are MLDv2 Reports with INCLUDE filter mode and NONE as a source list. Using such messages (MLDv2 Reports with INCLUDE:NONE record), a node instructs the MLDv2 capable routers that for the reported multicast address is interested in NONE address as a source address.

The version of a Multicast Listener Query message is determined as follows:

MLDv1 Query: length = 24 octets

MLDv2 Query: length >= 28 octets

Query messages that do not match any of the above conditions (e.g., a Query of length 26 octets) must be silently ignored.

In order to be compatible with MLDv1 routers, MLDv2 hosts must operate in version 1 compatibility mode when they receive MLDv1 Queries.

### 2.1.3 Security Measures Regarding MLD Messages

All MLD messages must be sent:

- With a link-local IPv6 Source Address.
- An IPv6 Hop Limit equal to 1, and
- An IPv6 Router Alert option (RFC 2711) in a Hop-by-Hop Options header. The Router Alert option is necessary to cause routers to examine MLD messages sent to IPv6 multicast addresses in which the routers themselves have no interest.

Unrecognized message types must be silently ignored.

## 2.2 Queriers

A router may perform both parts of the protocol, including sending Queries and even responding to its own messages. On the contrary, hosts can only be multicast listeners.

An MLD router sends periodical or triggered Query messages on the local subnet. If there is more than one MLD router, a Querier election mechanism is used to select a single MLD router to be in Querier state. This router is called the *Querier*. All multicast routers on the subnet listen to the messages sent by multicast address listeners, and maintain the same multicast listening information state, so that they can take over the Querier role in case the current Querier fails. Still, at any given time interval only the Querier sends periodical or triggered query messages on the subnet.

When a router starts operating on a subnet, by default it considers itself as being the Querier. Hence, it sends several General Queries separated by a small time interval. When a router receives a Query with a lower IPv6 address than its own, it switches to Non-Querier state and ceases to send queries on the link. However, if it stops receiving Queries for a specific amount of time, it re-enters the Querier state and begins sending General Queries again.

When an MLDv1 Done or an MLDv2 Report (INCLUDE:NONE) message is received, the Querier sends a last Query to check whether there are still listeners on that link for the specific multicast address (this is the so called "Last Listener

Query"). An MLDv1 Report can be omitted by a host once in the meantime an MLDv1 Request for the same multicast address has been received by another host. Similarly, a host may allow its MLDv2 Multicast Listener Report to be suppressed by an MLDv1 Report.

# 3 MLD Testing Categories

The MLD tests of the MLD capable components can be separated in the following basic categories:

- Compatibility tests
- Link-local security tests
- RFC compliance tests
- Resource consumption
- Denial of Service (DoS) using flooding and amplification attacks
- Denial of Service (DoS) attacks by abusing the protocol operation.
- Fuzzing
- Affecting the Neighbor Discovery process

Based on these categories, specific tests for the MLD protocol are described in this section. Some of the tests are suitable for either MLDv1 or MLDv2 only, while some other are common for both versions. For a better understanding of the tests please check the background theory in Section 3, as well as the format of the MLD messages in Appendix B.

The following list is by no means an exhaustive one of all possible tests. However, it can serve as a good basis for testing MLD capable devices.

## 3.1 Preparation – Checking the Default Values of the Systems

The default or used values of the MLD nodes can be checked either by looking at their corresponding configuration, or by performing simple tests. The values of the parameters that can be checked are the following:

- Robustness variable
- Query interval
- Query Response Interval
- Last Listener Query Interval
- Unsolicited Report Interval

For more information regarding the usage of these parameters please see RFC 2710 and RFC 3810.

## 3.2 Compatibility Tests

These tests are common for both versions of the MLD protocol and they can be summarized as following:

- Does a specific OS accept MLDv1 Queries?
  - How does it respond to them? Using MLDv1 or MLDv2 Reports?
- Are there MLDv1-only capable routers (or can they be configured to MLDv1 only)?
  - If yes, how do they react to MLDv2 Reports?
- Which routers accept MLDv1 Report/Done messages?
- How do MLDv2 routers respond to MLDv1 Queries and Done messages? Do they send MLDv1 Queries or MLDv2 ones?
- How MLDv1/v2 capable nodes (routers/hosts) behave in a mixed environment?

## 3.3 Link-Local Security Tests

These tests are also common for both versions of the MLD protocol and they can be summarized as following:

- Do the MLD nodes (routers, hosts) accept MLD messages of any type or version when:
  - The Hop-Limit is not equal to 1?
  - The IPv6 source address is not a link-local one?

## 3.4 RFC Compliance Tests

### 3.4.1 For both MLDv1 and MLDv2

- Do the MLD nodes (routers, hosts) accept MLD messages of any type or version when there is no a Router alert Option in a Hop-by-Hop Extension header (especially in MLDv1/v2 Report and MLDv1 Done messages)?
  - If such messages are accepted by hosts, routers may not be alerted/informed. In such a case, we may be able to interact with the hosts without the legitimate router to be aware of it. From an attacker's perspective, this can be helpful for example to achieve MLD Report suppression, if supported by the target.

### 3.4.2 For MLDv1 Only

- What if multicast addresses or link-local addresses (but non-unicast ones) are included as sources in MLD Reports/Done messages? Combine with scenarios where MLD Snooping is in place.

### 3.4.3 For MLDv2 Only

- What if multicast addresses or link-local addresses (but non-unicast ones) are included as sources in INCLUDE/EXCLUDE Reports? Combine with MLD Snooping.
- Send spoofed Current State Reports to check if Queries are triggered – they shouldn't.
  - If yes, it can be (ab-) used for flooding/amplification attacks.
- Repeat the same as above, but with a different state than the legitimate one. Are these Current State Reports processed?
  - If yes, this can be (ab-) used for flooding/amplification attacks.
- Can we influence the Robustness variable at other routers if we temporarily take-over the Querier Role and use a maximum Robustness variable?

## 3.5 Resource Consumption Tests

### 3.5.1 For MLDv1 Only

- Fill router's buffer with multicast address listeners using too many MLD Reports → try to overflow it.
- Increase the CPU load at routers by flooding with Report/Done messages.

### 3.5.2 For MLDv2 Only

- Try to flood with forged State Change Report message to increase processing on each router and on each listener of the multicast address.
- Use "State Change Reports" to increase traffic and router load by changing states (INCLUDE/EXCLUDE – Types 3 and 4) continuously for many multicast addresses.
- Send EXCLUDE messages with many multicast addresses to fill the states/buffers at router. Flood the targets with such messages.
- Use Type 5 or Type 6 to add more sources in INCLUDE/EXCLUDE to fill the buffer of routers without having to send all the sources in one MLD Report message. (variation of the previous test).

## 3.6 Denial of Service Attacks by Using Flooding and Amplification Attacks

### 3.6.1 For MLDv1 Only

- Flood the network using spoofed Queries, setting the Maximum Response Delay parameter equal to 0).
- Flood the network using spoofed Queries, setting the Maximum Response Delay parameter equal to 0 and maximizing the Robustness variable.
    - Do systems (e.g. Windows) respond with more than two (2) Reports per Query?
- Flood the network using spoofed Done messages for ("reported") multicast addresses (those should trigger Last Listener Queries and consequently, Reports from the listeners).
    - Repeat the above test using FF02::1 as a destination address. This may trigger Queries and Reports for all nodes in the link-local.

### 3.6.2 For MLDv2 Only

Check if OSs respond with more than one Report message per Query (they shouldn't). Perform the checks for all the types of Queries.

- Check if maximizing the Robustness variable at Queries affects the number of Reports triggered by Queries (it shouldn't).
- Flood with "State Change Report" messages (INCLUDE:NONE for existing listeners) to trigger Multicast Address Specific Queries and Multicast Address and Source Specific Queries (they should trigger a Last Listener Query and then Report for legitimate users).
    (a) Perform this continuously
    (b) Perform this for all listeners.
    To which extent is the traffic increased?
- Send MLD Queries/Reports using as many sources/multicast addresses as possible while fragmenting the packet to achieve the maximum possible IPv6 datagram(s). Then, flood the network with such packets.
- Similarly to the previous test, but add arbitrary additional data. In MLD Reports, also add Auxiliary data.

- Check if by increasing the Robustness variable, State Change Reports are increased. If yes, it will generate much traffic in a Windows environments due to their potential "auto-changing" behavior.

## 3.7 Denial of Service Attacks by Abusing the Protocol Operation

### 3.7.1 For Both MLDv1 and MLDv2

- Spoof a Querier with a numerically lower IPv6 address and cease operation for a time.
    - Do other routers take over the Querier role after the specified interval?
- Check if a Query sent to a router's unicast or link-local address (and not to FF02::1) make it believe that a Query has been sent to the link.
        - If yes, combine it with the previous test (the legitimate router shouldn't take over the Querier role again).
        - In MLDv2, what happens if the Queries are sent to ff02::16 (all MLDv2 capable routers instead of all nodes)?

### 3.7.2 For MLDv1 Only

- Send spoofed Multicast Done messages for listening addresses. Can you trigger "Last Listener Queries" or remove a multicast address ("Listener entry") from a router?
    - Usually this should work for "normal" multicast traffic. We must have taken over the Querier state before to suppress the "last Query" message.
    - It can also be combined with MLD Snooping enabled in switches.
- Multicast Done messages: Check what happens if the multicast address field is filled with a "generic" multicast address (e.g. FF02::1). Can you "cease" any operation (routers or hosts to stop/prevent from listening on such a multicast address)?
    - It could be effective when MLD Snooping is enabled in switches.
- Can we send a spoofed Done message to the Router's address (link-local, unicast, or FF02::2) using as source address the link-local address of the router itself? Can we make it to stop listening to FF02::16 or FF02::2?
    - Try to exploit the fact that, according to the RFC, "A router may perform both parts of the protocol, including responding to its own message".
- Exploit duplicate Reports suppression by sending faked – not necessarily spoofed – MLD Reports to specific unicast IPv6 addresses → make them believe that a Report has been sent.
    - To be effective, targets must accept Reports when the destination address of MLD Reports is not a multicast one, at least NOT the "all nodes" one.
    - It should work if an OS accepts MLD messages without a Router Alert option.
- Become a Querier and then send spoofed Done messages. Routers in a non-Querier state MUST not send Last Listener Queries.
    - DoS can potentially be achieved if a non-Querier router accepts Multicast Address Specific Queries to addresses other than link-local all-nodes, e.g. by using just the Router-Alert (otherwise nodes will respond with Report messages).
- Send Queries with setting a really big Maximum Response Delay parameter, e.g. >> 125 seconds (default value of query intervals). Can we actually make the hosts not to send their Reports?

### 3.7.3 For MLDv2 Only

- Take over the Querier role, spoof INCLUDE None messages to FF02::16 and then send a Multicast Address and Source Specific Query to the unicast addresses or FF02:16 of the non-Querier routers: Implicit DoS.
    - Test the above in scenarios with MLD Snooping enabled.
- MLDv2 Report messages using INCLUDE:NONE mode/filtering: Check what happens if the multicast address field is filled with a "generic" multicast address (e.g. FF02::1). Can you "cease" any operation (routers or hosts to stop listening on such a multicast address)? Could be effective when MLD snooping present.
- Can we send a spoofed MLDv2 Report message to the Router (either link-local, unicast, or FF02::2) using as source add. the link-local address of the router itself? Can we make it to stop listening to FF02::16/ FF02::2?
- Send Queries by setting a really big Maximum Response Code parameter, e.g. >> 125 seconds (default value of query intervals). Can we actually make the hosts not to send their Reports?

## 3.8 Affecting the Neighbor Discovery Process

The Neighbor Discovery process uses solicited-nodes multicast addresses. MLD snooping implemented by some vendors may filter traffic based on the solicited-node groups. So, a SLAAC attack which puts an attacker in the middle (e.g. using parasite6 of the thc ipv6 attacking toolkit) may not be effective in such environments. In this scenario, it is examined if you can overcome the MLD Snooping restrictions, when exist regarding the solicited-node multicast addresses and hence, to perform the usual ND SLAAC attack and put yourself in the middle.

### 3.8.1 For MLDv1 Only

- Use spoofed Done messages to DoS clients implicitly. Must be combined with MLD Snooping enabled in switches.
- Use spoofed MLDv1 Report messages to "steal" traffic / put you in the middle of the group. Must be combined with MLD Snooping enabled in switches.

### 3.8.2 For MLDv2 Only

- Use spoofed MLDv2 Report (INCLUDE:NONE) messages to DoS clients implicitly. Must be combined with MLD Snooping enabled in switches.
- Use spoofed MLDv2 Report messages to "steal" traffic / put you in the middle of the group. Must be combined with MLD Snooping enabled in switches.

## 3.9 Fuzzing

### 3.9.1 For MLDv1 messages

- Create and send MLDv1 Queries – Reports – Done messages which are much bigger than 24 octets of bytes.
- Set the Maximum Response Delay values in Report and Done messages ≠ 0.

### 3.9.2 For MLDv2 Messages

- Try unrecognized/unknown Record Types. Type 0?
- Send MLDv2 Reports with Record Type 1 and empty source list.

- Send Multicast Address and Source Specific Queries with many sources but with wrong (zero or not) number of sources.
- Send MLD Reports where the number of Multicast Records is smaller than the actual number of Multicast Records.

In Appendix A specific examples regarding the fuzzing tests are given using the Chiron tool (briefly described in the next section).

# 4 THE MAIN TOOL FOR PERFORMING MLD TESTS

The most suitable tool for performing the described tests is Chiron (it can be downloaded from *http://www.secfu.net/tools-scripts/*). Chiron is an IPv6 penetration testing framework which incorporates several modules, the most useful of which for our purposes is the *link-local one*. The aforementioned module makes it possible for the user to craft arbitrary IPv6 MLD messages. The most relevant options for crafting arbitrary MLD packets are listed below.

| Chiron Option | What it Does |
|---|---|
| -d <IPv6 address> | Sets the destination address field in the IPv6 header of the packet to the address specified. |
| -s <IPv6 address> | Sets the source address field in the IPv6 header of the packet to the provided address. |
| -m <MAC address> | Allows the user to specify the source MAC address to be put in the Ethernet header of the final packet. |
| -tm <MAC address> | Defines the destination MAC address of the final packet. |
| -ralert | Puts an *RTR-ALERT* extension header after the IPv6 header. |
| -mldv1q | Chiron sets the payload for the generated IPv6 datagram to an MLDv1 *Query*. |
| -mldv1d | Causes Chiron to use an MLDv1 *Done* message as payload for the generated IPv6 datagram. |
| -mul_addr <IPv6 address> | When crafting MLDv1 messages, set the *Multicast-Address* field in the corresponding *Query*, *Report* or *Done* messages to the provided multicast address. |
| -mldmrd <milliseconds> | For MLDv1 and MLDv2 *Queries*, Chiron sets the *Maximum Response Delay* field in the MLDv1 header to the value specified. |
| -qrv <int> | Allows the user to define the value of the *Query Robustness Variable* in an MLDv1 or MLDv2 *Query*. |
| -mldv2q | Chiron uses an MLDv2 *Query* as payload for the crafted IPv6 datagram. |
| -mldv2r | Causes Chiron to set the payload of the generated IPv6 datagram to an MLDv2 Report. |
| -mldv2rm | When crafting MLDv2 *Reports*, allows the user to send several independent *Reports* each containing a single *Multicast Address Record* in a range of IPv6 addresses. |
| -no_of_mult_addr_recs <int> | Sets the number of *Multicast Address Records* in an MLDv2 *Report* to the value specified. |
| -lmar "(...)" | Allows the user to provide a list which defines the characteristics of the *Multicast Address Records* contained in an MLDv2 *Report*. |

| Chiron Option | What it Does |
|---|---|
| -mldv2rmo | When crafting MLDv2 *Reports*, allows the user to send several *Multicast Address Records* for a range of IPv6 addresses all in the same *Report*. |
| -mldv2rms | When crafting MLDv2 reports, allows the user to define Multicast Address Records whose list of sources is generated based on a provided range. |
| -nf <number of fragments> | Chiron fragments the resulting packet into the number of fragments specified by the user. |
| -fl | Puts Chiron in flooding mode. |
| -flooding_interval <seconds> | When in flooding mode, causes Chiron to craft and send the packet defined by the options and parameters provided periodically after the number of seconds defined have passed. |

*Table 1: Chiron parameters for crafting arbitrary IPv6 MLD packets*

A simple example of crafting MLDv2 Queries is displayed below:

```
./chiron_local_link.py eth0 –mldv2q -ralert -mldmrd 0
```

The command listed above instructs the local-link module of Chiron to craft and MLDv2 *Query* with its *Maximum-Response-Delay* value set to zero and send it through the network interface "`eth0`".

The following command presents a slightly more complicated example of a Chiron command for sending an MLDv2 Reports.

```
./chiron_local_link.py eth0 -mldv2rmo -luE 0'(options=RouterAlert)' -no_of_mult_addr_recs 1024 -res 3 -lmar "(rtype=4;dst=ff08::2000-2400)" -nf 60
```

The command shown above sends a fragmented MLDv2 Report using the interface eth0. This report contains 800 Multicast Address Records (MARs) with a record type value of 4, while the Reserved field is set to 3. The Report is filled with MARs using the addresses from ff08::2000 to ff08::2400. Due to its huge size, this report is sent in sixty (60) fragments.

The command below sends twenty MLDv2 *Reports* each one with a specific source.

```
./chiron_local_link.py eth0 -mldv2rm -ralert -no_of_mult_addr_recs 20 -lmar "(rtype=4;dst=ff08::1000-1014;no_of_sources=1;saddresses=2001:db8:1::a)"
```

However, changing "`-mldv2rm`" to "`-mldv2rmo`" leads to a single MLDv2 Report containing twenty MARs being sent. For more information, please check the Chiron manual.

# 5 Appendix A: Examples of Chiron Commands for MLD Fuzzing Tests

In this appendix some specific examples describing how to perform the fuzzing tests are given. The used tool for these test sis Chiron, described in the previous section. These examples can also be used as a basis for creating your own Chiron commands to reproduce the rests of the described tests, or even other tests that the reader has created.

In each case, please substitute *$iface* with your interface (e.g. eth0).

Enjoy!

## 5.1 MLDv1

Before we start: Force the usage of MLDv1 by sending periodic MLDv1 Generic Queries.

```
./chiron_local_link.py $iface -mldv1q -ralert -d ff02::1 -fl -flooding-interval 50
```

1. Send MLDv1 Queries – Reports – Done messages >> 24 octets of bytes. If datagram > MTU, must be fragmented -> Use Chiron

Maximum possible: 65535 bytes after reassembly

```
./chiron_local_link.py $iface -mldv1d -ralert -d ff02::1 -l4_data `python -c 'print "EEFFGGH"+"AABBCCDD" * 8187'` -mul_addr "AAAA:BBBB:CCCC:DDDD:EEEE:FFFF:0000:1111" -nf 60 -mldmrd 65535

./chiron_local_link.py $iface -mldv1r -ralert -d ff02::1 -l4_data `python -c 'print "EEFFGGH"+"AABBCCDD" * 8187'` -mul_addr  "AAAA:BBBB:CCCC:DDDD:EEEE:FFFF:0000:1111" -nf 60 -mldmrd 65535

./chiron_local_link.py $iface -mldv1q -ralert -d ff02::1 -l4_data `python -c 'print "EEFFGGH"+"AABBCCDD" * 8187'` -mul_addr "AAAA:BBBB:CCCC:DDDD:EEEE:FFFF:0000:1111" -nf 60 -mldmrd 65535
```

2. Send MLDv1 Queries – Reports – Done messages with the Maximum Response Delay value fuzzed or set to the maximum possible one (has been found in the past that this caused problems – CVE-2008-2464).

```
./chiron_local_link.py $iface -mldv1q -ralert -d ff02::1 -mldmrd 65535

./chiron_local_link.py $iface -mldv1r -ralert -d ff02::1 -mldmrd 65535

./chiron_local_link.py $iface -mldv1d -ralert -d ff02::1 -mldmrd 65535

./chiron_local_link.py $iface -mldv1q -ralert -d ff02::1 -mldmrd 32768

./chiron_local_link.py $iface -mldv1q -ralert -d ff02::1 -mldmrd 32767

./chiron_local_link.py $iface -mldv1r -ralert -d ff02::1 -mldmrd 32767

./chiron_local_link.py $iface -mldv1r -ralert -d ff02::1 -mldmrd 32768
```

```
./chiron_local_link.py $iface -mldv1d -ralert -d ff02::1 -mldmrd 32767

./chiron_local_link.py $iface -mldv1d -ralert -d ff02::1 -mldmrd 32768
```

## 5.2 MLDv2

### 5.2.1 Queries

1. Fuzz the following parameters to MLDv2 Queries:

   a. Maximum Response Code

min – middle – max: 0 – 32767/32768 – 65535

```
./chiron_local_link.py $iface -mldv2q -ralert -d ff02::1 -mldmrd 65535

./chiron_local_link.py $iface -mldv2q -ralert -d ff02::1 -mldmrd 32768

./chiron_local_link.py $iface -mldv2q -ralert -d ff02::1 -mldmrd 32767

./chiron_local_link.py $iface -mldv2r -ralert -d ff02::1 -mldmrd 65535

./chiron_local_link.py $iface -mldv2r -ralert -d ff02::1 -mldmrd 32768

./chiron_local_link.py $iface -mldv2r -ralert -d ff02::1 -mldmrd 32767
```

   b. QRV: Values 0-7

   c. QQIC: QQIC: 0 – 127/128 – 255

```
./chiron_local_link.py $iface -mldv2q -ralert -d ff02::1 -mldmrd 65535 -qqic 255 -qrv 7
./chiron_local_link.py $iface -mldv2q -ralert -d ff02::1 -mldmrd 65535 -qqic 127 -qrv 7
./chiron_local_link.py $iface -mldv2q -ralert -d ff02::1 -mldmrd 65535 -qqic 128 -qrv 7
./chiron_local_link.py $iface -mldv2r -ralert -d ff02::1 -mldmrd 65535 -qqic 255 -qrv 7
./chiron_local_link.py $iface -mldv2r -ralert -d ff02::1 -mldmrd 65535 -qqic 127 -qrv 7
./chiron_local_link.py $iface -mldv2r -ralert -d ff02::1 -mldmrd 65535 -qqic 128 -qrv 7
```

2. Send arbitrary huge MLDv2 datagram: Send Generic Queries (sent to ff02::1) with number of sources =0 or > 0 and many source addresses. Add as many arbitrary data as possible at the end of an MLD Query. Try to achieve as close to the real Maximum Number of Sources as possible.

```
./chiron_local_link.py $iface -mldv2q -ralert -d ff02::1 -l4_data `python -c 'print "AABBCCDD" * 8187'` -mul_addr
"AAAA:BBBB:CCCC:DDDD:EEEE:FFFF:0000:1111" -nf 60 -mldmrd 65535  -qqic 255 -qrv 7 -no_of_sources 511

./chiron_local_link.py $iface -mldv2q -ralert -d ff02::1 -l4_data `python -c 'print "AABBCCDD" * 8187'` -mul_addr
"AAAA:BBBB:CCCC:DDDD:EEEE:FFFF:0000:1111" -nf 60 -mldmrd 65535  -qqic 255 -qrv 7 -no_of_sources 0
```

```
./chiron_local_link.py $iface -mldv2q -ralert -d ff02::1 -l4_data `python -c 'print "AABBCCDD" * 8187'` -mul_addr "AAAA:BBBB:CCCC:DDDD:EEEE:FFFF:0000:1111" -nf 60 -mldmrd 65535  -qqic 255 -qrv 7 -no_of_sources 65535

./chiron_local_link.py $iface -mldv2q -ralert -d ff02::1 -l4_data `python -c 'print "EEF"+"AABBCCDD" * 8187'` -mul_addr "AAAA:BBBB:CCCC:DDDD:EEEE:FFFF:0000:1111" -nf 60 -mldmrd 65535  -qqic 255 -qrv 7 -no_of_sources 65535

./chiron_local_link.py $iface -mldv2q -ralert -d ff02::1 -l4_data `python -c 'print "EEF"+"AABBCCDD" * 8187'` -mul_addr "AAAA:BBBB:CCCC:DDDD:EEEE:FFFF:0000:1111" -nf 60 -mldmrd 65535  -qqic 255 -qrv 7 -no_of_sources 511

./chiron_local_link.py $iface -mldv2q -ralert -d ff02::1 -l4_data `python -c 'print "EEF"+"AABBCCDD" * 8187'` -mul_addr "AAAA:BBBB:CCCC:DDDD:EEEE:FFFF:0000:1111" -nf 60 -mldmrd 65535  -qqic 255 -qrv 7 -no_of_sources 0
```

3. Maximize the Number of Sources and use zero or very few real sources.

```
./chiron_local_link.py $iface -mldv2q -ralert -d ff02::1 -mul_addr "AAAA:BBBB:CCCC:DDDD:EEEE:FFFF:0000:1111" -mldmrd 65535  -qqic 255 -qrv 7 -no_of_sources 65535

./chiron_local_link.py $iface -mldv2q -ralert -d ff02::1 -mul_addr "AAAA:BBBB:CCCC:DDDD:EEEE:FFFF:0000:1111" -mldmrd 65535  -qqic 255 -qrv 7 -no_of_sources 32767

./chiron_local_link.py $iface -mldv2q -ralert -d ff02::1 -mul_addr "AAAA:BBBB:CCCC:DDDD:EEEE:FFFF:0000:1111" -mldmrd 65535  -qqic 255 -qrv 7 -no_of_sources 32768

./chiron_local_link.py $iface -mldv2q -ralert -d ff02::1 -mul_addr "AAAA:BBBB:CCCC:DDDD:EEEE:FFFF:0000:1111" -mldmrd 65535  -qqic 255 -qrv 7 -no_of_sources 16383

./chiron_local_link.py $iface -mldv2q -ralert -d ff02::1 -mul_addr "AAAA:BBBB:CCCC:DDDD:EEEE:FFFF:0000:1111" -mldmrd 65535  -qqic 255 -qrv 7 -no_of_sources 16384
```

Maximize the Number of Sources and use very few real sources.

```
./chiron_local_link.py $iface -mldv2q -ralert -d ff02::1 -l4_data `python -c 'print "AABBCCDD" * 2'` -mul_addr "AAAA:BBBB:CCCC:DDDD:EEEE:FFFF:0000:1111" -mldmrd 65535  -qqic 255 -qrv 7 -no_of_sources 65535

./chiron_local_link.py $iface -mldv2q -ralert -d ff02::1 -l4_data `python -c 'print "AABBCCDD" * 2'` -mul_addr "AAAA:BBBB:CCCC:DDDD:EEEE:FFFF:0000:1111" -mldmrd 65535  -qqic 255 -qrv 7 -no_of_sources 32767

./chiron_local_link.py $iface -mldv2q -ralert -d ff02::1 -l4_data `python -c 'print "AABBCCDD" * 2'` -mul_addr "AAAA:BBBB:CCCC:DDDD:EEEE:FFFF:0000:1111" -mldmrd 65535  -qqic 255 -qrv 7 -no_of_sources 32768

./chiron_local_link.py $iface -mldv2q -ralert -d ff02::1 -l4_data `python -c 'print "AABBCCDD" * 2'` -mul_addr "AAAA:BBBB:CCCC:DDDD:EEEE:FFFF:0000:1111" -mldmrd 65535  -qqic 255 -qrv 7 -no_of_sources 16383
```

```
./chiron_local_link.py $iface -mldv2q -ralert -d ff02::1 -l4_data `python -c 'print "AABBCCDD" * 2'` -mul_addr
"AAAA:BBBB:CCCC:DDDD:EEEE:FFFF:0000:1111" -mldmrd 65535  -qqic 255 -qrv 7 -no_of_sources 16384

./chiron_local_link.py $iface -mldv2q -ralert -d ff02::1 -l4_data `python -c 'print "AABBCCDD" * 4'` -mul_addr
"AAAA:BBBB:CCCC:DDDD:EEEE:FFFF:0000:1111" -mldmrd 65535  -qqic 255 -qrv 7 -no_of_sources 65535

./chiron_local_link.py $iface -mldv2q -ralert -d ff02::1 -l4_data `python -c 'print "AABBCCDD" * 4'` -mul_addr
"AAAA:BBBB:CCCC:DDDD:EEEE:FFFF:0000:1111" -mldmrd 65535  -qqic 255 -qrv 7 -no_of_sources 32767

./chiron_local_link.py $iface -mldv2q -ralert -d ff02::1 -l4_data `python -c 'print "AABBCCDD" * 4'` -mul_addr
"AAAA:BBBB:CCCC:DDDD:EEEE:FFFF:0000:1111" -mldmrd 65535  -qqic 255 -qrv 7 -no_of_sources 32768

./chiron_local_link.py $iface -mldv2q -ralert -d ff02::1 -l4_data `python -c 'print "AABBCCDD" * 4'` -mul_addr
"AAAA:BBBB:CCCC:DDDD:EEEE:FFFF:0000:1111" -mldmrd 65535  -qqic 255 -qrv 7 -no_of_sources 16383
./chiron_local_link.py $iface -mldv2q -ralert -d ff02::1 -l4_data `python -c 'print "AABBCCDD" * 4'` -mul_addr
"AAAA:BBBB:CCCC:DDDD:EEEE:FFFF:0000:1111" -mldmrd 65535  -qqic 255 -qrv 7 -no_of_sources 16384

./chiron_local_link.py $iface -mldv2q -ralert -d ff02::1 -mul_addr "AAAA:BBBB:CCCC:DDDD:EEEE:FFFF:0000:1111" -
mldmrd 65535  -qqic 255 -qrv 7 -no_of_sources 65535

./chiron_local_link.py $iface -mldv2q -ralert -d ff02::1 -mul_addr "AAAA:BBBB:CCCC:DDDD:EEEE:FFFF:0000:1111" -
mldmrd 65535  -qqic 255 -qrv 7 -no_of_sources 32767

./chiron_local_link.py $iface -mldv2q -ralert -d ff02::1 -mul_addr "AAAA:BBBB:CCCC:DDDD:EEEE:FFFF:0000:1111" -
mldmrd 65535  -qqic 255 -qrv 7 -no_of_sources 32768

./chiron_local_link.py $iface -mldv2q -ralert -d ff02::1 -mul_addr "AAAA:BBBB:CCCC:DDDD:EEEE:FFFF:0000:1111" -
mldmrd 65535  -qqic 255 -qrv 7 -no_of_sources 16383

./chiron_local_link.py $iface -mldv2q -ralert -d ff02::1 -mul_addr "AAAA:BBBB:CCCC:DDDD:EEEE:FFFF:0000:1111" -
mldmrd 65535  -qqic 255 -qrv 7 -no_of_sources 16384
```

### 5.2.2 Reports

4. Fuzz Record Types

```
for record_type in `seq 0 255`
do
        ./chiron_local_link.py $iface -mldv2r -ralert -no_of_mult_addr_recs 1 -res 3 -res 5 -lmar
"(rtype=$record_type;dst=ff15::38;no_of_sources=2;saddresses=ff02::3-ff02::4;auxdata='AAAA';auxdatalen=1)"
done
```

5. Send MLDv2 Reports with Record Type 1 and an empty source list.

```
./chiron_local_link.py $iface -mldv2r -ralert -no_of_mult_addr_recs 1 -res 3 -res 5 -lmar
"(rtype=1;dst=ff15::38;no_of_sources=0)"
```

```
./chiron_local_link.py $iface -mldv2r -ralert -no_of_mult_addr_recs 1 -res 3 -res 5 -lmar
"(rtype=1;dst=ff15::38;no_of_sources=0;auxdata='AAAA';auxdatalen=1)"
```

6. Send MLD Reports were Number of Multicast Records is smaller than the actual number of Multicast Records. Use as big list of Multicast Records as possible. If required, fragment it.
7. Maximize the number of Multicast Address Records and include zero or very few of them.

```
./chiron_local_link.py $iface -mldv2r -ralert -no_of_mult_addr_recs 65535 -res 3 -res 5 -lmar `python -c 'print
"(rtype=4;dst=ff15::38;no_of_sources=0)," * 2'`"(rtype=4;dst=ff15::39;no_of_sources=0)" -nf 3

./chiron_local_link.py $iface -mldv2r -ralert -no_of_mult_addr_recs 65535 -res 3 -res 5
```

8. Add very lengthy Auxiliary data to multicast address records. Fragment the datagrams, if required.

```
./chiron_local_link.py $iface -mldv2r -ralert -no_of_mult_addr_recs 1 -res 3 -res 5 -lmar
"(rtype=4;dst=ff15::38;no_of_sources=0;auxdata='AAAA';auxdatalen=4)"
```

The following is the maximum legitimate one per multicast address record

```
./chiron_local_link.py $iface -mldv2r -ralert -no_of_mult_addr_recs 1 -res 3 -res 5 -lmar
"(rtype=4;dst=ff15::38;no_of_sources=0;auxdatalen=255)" -l4_data `python -c 'print "A" * 1020'`

./chiron_local_link.py $iface -mldv2r -ralert -no_of_mult_addr_recs 1 -res 3 -res 5 -lmar
"(rtype=4;dst=ff15::38;no_of_sources=0;auxdatalen=255)" -l4_data `python -c 'print "A" * 10'`

./chiron_local_link.py $iface -mldv2r -ralert -no_of_mult_addr_recs 1 -res 3 -res 5 -lmar
"(rtype=4;dst=ff15::38;no_of_sources=0;auxdatalen=255)"

./chiron_local_link.py $iface -mldv2r -ralert -no_of_mult_addr_recs 1 -res 3 -res 5 -lmar
"(rtype=4;dst=ff15::38;no_of_sources=0;auxdatalen=0)" -l4_data `python -c 'print "A" * 1020'`

./chiron_local_link.py $iface -mldv2r -ralert -no_of_mult_addr_recs 1 -res 3 -res 5 -lmar
"(rtype=4;dst=ff15::38;no_of_sources=0;auxdatalen=1)" -l4_data `python -c 'print "A" * 1020'`

./chiron_local_link.py $iface -mldv2r -ralert -no_of_mult_addr_recs 1 -res 3 -res 5 -lmar
"(rtype=4;dst=ff15::38;no_of_sources=0;auxdatalen=255)" -l4_data `python -c 'print "EFG"+"AABBCCDD" * 8187'` -nf 60
```

# 6 APPENDIX B. FORMAT OF THE MLD MESSAGES

In this appendix, the format of the MLD messages as defined in RFC 2710 and RFC 3810 are presented. In each case, a brief description of the fields related with our tests is given. For more information regarding these fields or the rest of them not described here, please check the corresponding RFCs.

## 6.1 MLDv1

The MLD messages have a generic format, which is displayed below:

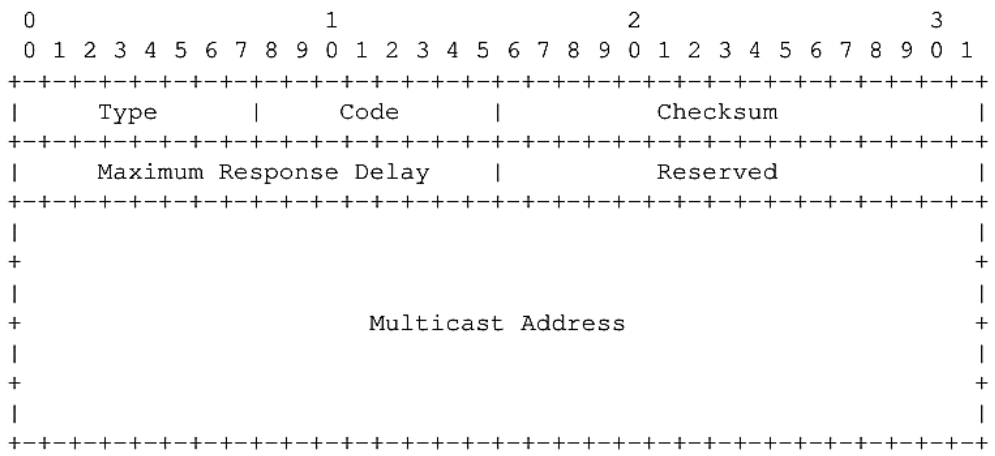


*Figure 1: The MLDv1 Messages Format (source: RFC 2710)*

Where:

**Type:** It is defined based on the MLD Message Type (130 for Query, 131 for Request, 132 for Done).

The **Maximum Response Delay (MRD)** is the maximum allowed delay (in milliseconds) before sending a responding Report. It allows tuning of the burstiness of the MLD traffic on the link. It also allows the routers to tune the "leave latency". Default: 10000 (10 sec). It is meaningful only in Queries (it is set to 0 in other messages).

The **Multicast Address**:

Is set to zero for General Queries.

Is set to a specific multicast address for Multicast Address Specific Queries.

Is set to a specific multicast address to which the sender is listening in Report message.

In a Done Message, it is set to a specific multicast address to which the sender stops listening.

## 6.2 MLDv2

MLDv2 Queries and Reports have different formats, while there are no MLDv2 Done messages.

The MLDv2 Query messages have the following format:

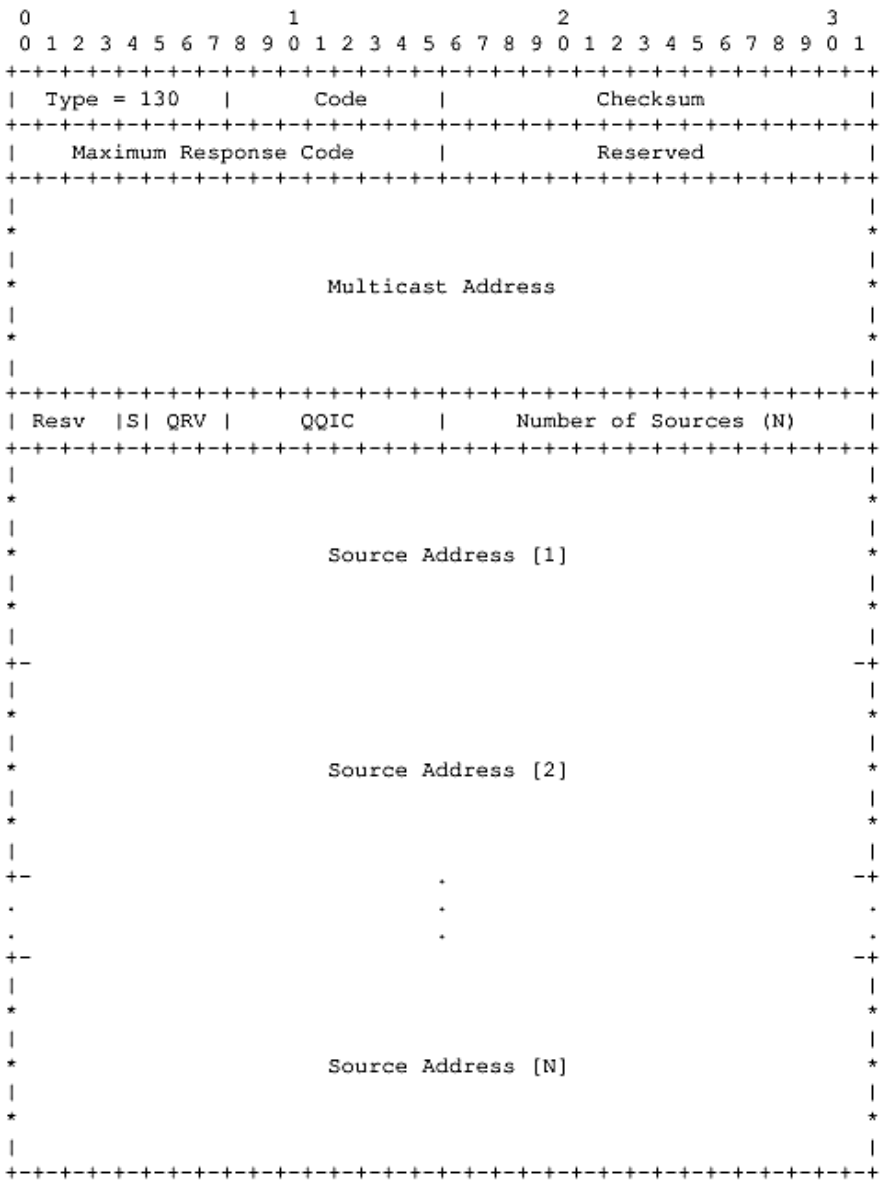


*Figure 2: The MLDv2 Query Messages Format (source: RFC 3810)*

Where:

The **Maximum Response Code** field specifies the maximum time allowed before sending a responding Report. The actual time allowed, called the Maximum Response Delay, is represented in units of milliseconds, and is derived from the Maximum Response Code using a formula. For more information, please check RFC 3810, page 16.

The **QRV** (**Querier's Robustness Variable**) contains the Robustness Variable value used by the Querier.

The MLDv2 Report messages have the following format:

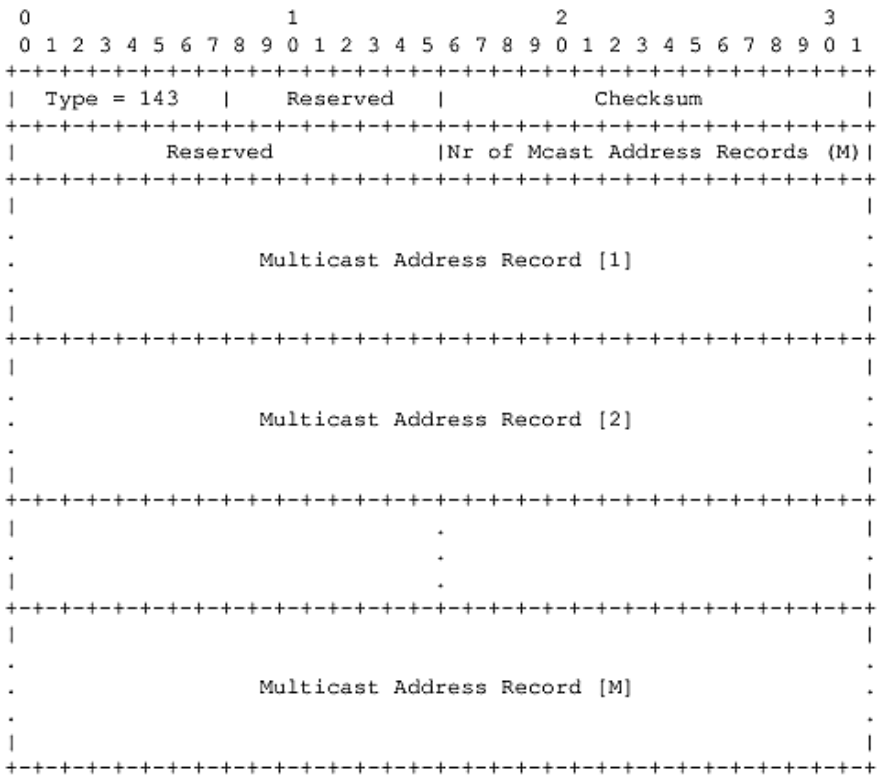


*Figure 3: The MLDv2 Report Messages Format (source: RFC 3810)*

Where:

**Nr of Mcast Address Records (M)** is the number of the Multicast Address Records included in the specific MLDv2 Report message.

Each Multicast Address Record has the following format:

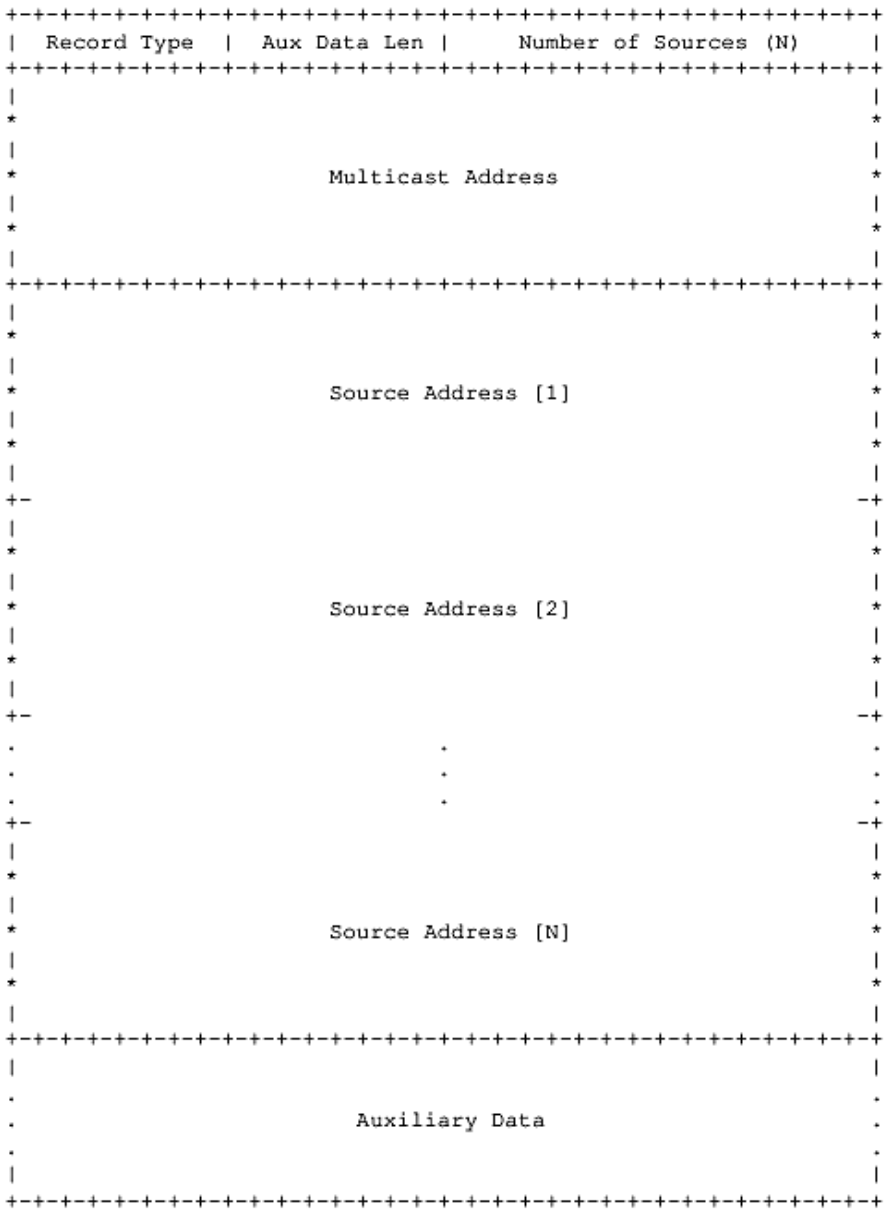


*Figure 4: The MLDv2 Multicast Address Records – Part of MLDv2 Report Messages (source: RFC 3810)*

Where:

The **Record Type** specifies the type of the Multicast Address Record.

The **Aux Data Len** field contains the length of the Auxiliary Data Field in this Multicast Address Record.

The **Number of Sources (N)** field specifies how many source addresses are present in this Multicast Address Record.

The **Multicast Address** field contains the multicast address to which this Multicast Address Record pertains.