Zero Conf Protocols and their numerous Man in the Middle (MITM) Attacks

Dhia Farrah, Marc Dacier
Digital Security Department
EURECOM
Sophia Antipolis, France
{dhia.farrah, marc.dacier}@eurecom.fr

***Abstract*—Zero conf protocols date from 1999. They provide plug and play mechanisms to set up networks without having to configure DNS or DHCP servers. Almost every device (PCs, printers, scanners, etc.) nowadays "speaks" one of these protocols, sometimes without its owner being even aware of it. The booming IoT ecosystem, in particular, relies heavily on them. Unfortunately, these protocols offer a number of different ways to run, so called, man in the middle attacks (MITM). Some previous publications have mentioned and have taken advantage of one or another of these design flaws. In this paper, we provide a deep dive into the various issues at hand and show the extent of the problem. We consider that the growing reliance of networks on these protocols represent an underestimated and ill covered threat. We have run a number of experiments (300) to test various implementations and discuss our results. We also propose means to detect these attacks thanks to Zeek (aka Bro). We make the attack code as well as the Zeek scripts available to the research community in a format that makes replication of our results possible by researchers while not easy to use by script kiddies.**

***Index Terms*—; MITM; LLMNR; zeroconf protocols**

I. Introduction

In 1999,the IETF chartered the Zeroconf working group to enable networking in the absence of configuration and administration. "*[...] The long-term goal of Zero Configuration Networking is to enable the creation of entirely new kinds of networked products, products that today would simply not be commercially viable because of the inconvenience and support costs involved in setting up, configuring, and maintaining a network to allow them to operate [...]*" [1]. In fact, the genesis of this work starts in 1986 with Appletalk [2]. Indeed, it is worth noting, as explained by Cheshire and Krochmal in [3], that one of the goals of this group was to provide an IP-based replacement for AppleTalk and the AppleTalk Name Binding Protocol (NBP) [4], [5]. That vision is now achieved with the explosion of the so called Internet of Things, with a myriad of devices that rely on these protocols to communicate together.

Several protocols fall under the umbrella of "zero-conf protocols". The most common ones are IPv4LL [6], LLMNR [7], multicast DNS (mDNS]) [8], DNS-based Service Discovery (DNS-SD) [9]. Apple has implemented these last two under the name Apple Bonjour [10] and they can be found under the name Avahi [11] within Linux machines.

These protocols enable devices to, among other things, obtain an IP address, make the services they offer visible, search for others, etc., without the need of a DNS or DHCP server. The key idea is to use multicast addresses to reach out to all the other devices in the same network and "negotiate" IP addresses, host or service names, that a given device would like to use or to offer.

There are security flaws with these protocols and some previous works have shown how to leverage them to launch attacks [12]–[14]. In this paper, we propose a deep dive into the issues related to the most popular of these protocols, namely mDNS and DNS-SD. In particular, we show the many different ways to run a man in the middle attack in such environment. We carry out a number of experiments against various target devices and discuss the outcomes. Last but not least, we discuss how to detect them thanks to Zeek (aka Bro), a freely available network based intrusion detection system [15]–[17].

It is important to note that, even in networks configured with a DNS and a DHCP server, these protocols can be running as well, rendering devices vulnerable to the attacks mentioned in this paper.

To discuss these topics, the paper is structured as follows. In section 2, for the sake of completeness, we rapidly present the mDNS and DNS-SD protocols. In section 3, we cover the related work and position our own contributions. In Section 4, we describe several attack scenarios and the different components impacting the way attacks can be carried out. In section 5, we present our various experiments and their visible consequences in the network. In Section 6, we discuss how detection can be carried out. Section 7 covers the ethical dimension of this work and Section 8 concludes the paper.

II. A brief introduction to mDNS and DNS-SD

A. Overview

This Section aims at making the paper self contained. We invite the reader already familiar with these protocols to move directly to Section III.

We start our explanations with the introduction of the key concepts and some important vocabulary (subsection

II-B). Then, we cover the three key phases of these protocols: the initiation phase (subsection II-C) run when a new device arrives in a network, the standby phase (subsection II-D) continuously run by existing devices and the conflict resolution phase (subsection II-E) run if two devices are claiming the same resource (e.g., an IP, a name, etc.).

B. Concepts and vocabulary

1) local domain name: In the context of zeroconf protocols, each device has a so called *local domain name*. The Fully Qualified Domain Name (FQDN) of this local domain name is made of the default host name, usually hardcoded in the device, concatenated with the suffix ".local". This suffix is a special Pseudo-Top-Level-Domain [18]. According to the RFC 6762 [8] "*... Any DNS query for a name ending with ".local." MUST be sent to the [...] multicast address 224.0.0.251 ...*" to be resolved by the mDNS protocol instead of querying any possible DNS server.

> **Example:** If a HP printer has "HP_XXX" as its hardcoded default name, then its *local domain name* will be "HP_XXX.local".

It is very important to note that each device can have, and usually has, a **general** domain name registered in the DNS server, when there is one, and a **local** domain name used by the mDNS protocol. The hostname part of these two FQDNs are usually the same but they can be different.

2) local service name: Each device can offer multiple services, such as a printing and a scanning service. Each service has a unique *local service name*. The format of this name is defined in [9] as "Instance.Service.Domain" where:

> **Domain:** is ".local";
> **Service:** follows the convention established in [9] to define a specific service (e.g., "_ipp._tcp" for a printing service);
> **Instance:** is hardcoded within the device and is different from the default hostname. To avoid conflicts, the last digits of the MAC address can, for instance, be added to it.
>
> **Example:** If a printer's name is "HP [D2A9BE]", then is local service name will be: "HP [D2A9BE] ._ipp._tcp.local".

If the device is offering several services, the same instance field value can be used in each local service instance but this is not mandatory (e.g., an Apple TV uses distinct local service instances values for the various local service names it announces).

C. Initiating phase

1) Obtaining an IP address: When a device joins a network, besides through a manual IP address configuration, it has two distinct ways to obtain an IP address:

1) DHCP server [19]: the device can search for a DHCP server, by broadcasting a request. If present, such server could offer an IP address to the device.
2) Automatic Private Internet Protocol Addressing: the device can randomly pick an IP address from the reserved address block 169.254.0.0/16. Then, it sends an Address Resolution Protocol (ARP) request [20] for that chosen IP. The absence of any reply indicates that no other device uses this IP and it is thus claimed by the device. In case of a reply, the device picks another IP and repeats the same process.

2) Binding Process: Once a device has obtained an IP address, it needs to verify that its *local domain name* as well as its *local service instance(s)* have not already been chosen by another device. This is achieved by **sending a query message** to the multicast address 224.0.0.251 which contains the local domain name and the local service instance(s). As with the ARP request, if the device receives no reply, it knows that these names are available and it then binds its IP to them. On the other hand, if another device replies, it means that the local domain name or one of the local service names is already taken and the conflict resolution process described later in II-E takes place.

Once the binding has taken place, the device sends to the multicast address several packets containing more information about the services it provides. Figure 1 offers a simplified schematic representation of this sequence of packets, where "HP 6263" is the printer's hostname and "HP Printer 123" the instance value of its local service name.

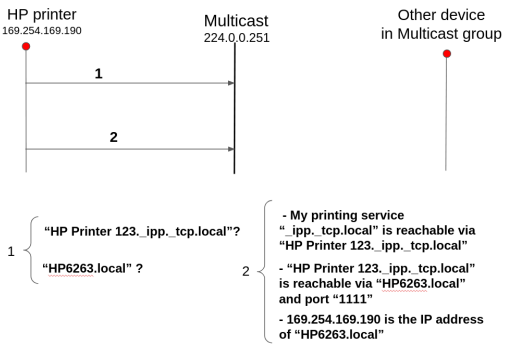

Fig. 1. Initiating Phase

D. Standby phase

After the initiating phase, each device keeps listening to the multicast address. Two types of messages can trigger a reaction:

1) a device is searching for devices providing a specific service, or is asking for all available services in the network. In such case, every concerned device replies with the requested information, usually (but not necessarily) to the multicast address instead of the unicast of the requestor. Such exchange of packets is represented in Figure 2;
2) another device, in its own initiating phase, is querying an already used local domain name or local service instance. In such a case, the conflict resolution process, described here after in II-E takes place.

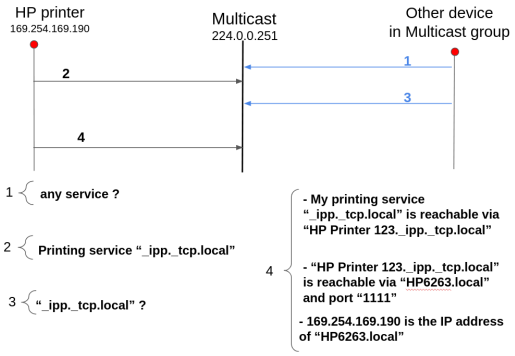

Fig. 2. Standby phase

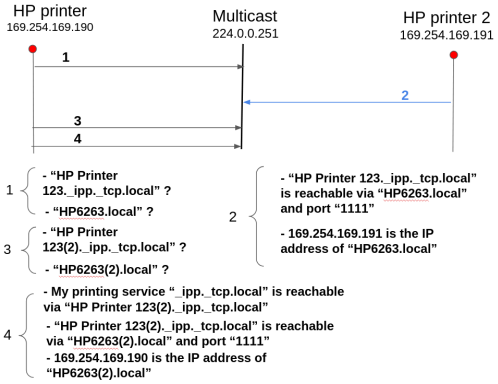

Fig. 4. Conflict with devices in distinct phases

E. Conflict resolution

This situation occurs when two devices want to use the same identical name (local domain name or local service instance). We can distinguish three different situations:

1) Both devices are in the initiating phase:
The two devices join the network at the same time. During their initiating phase, they can detect the conflict, as explained in Section 8.2, page 27, of the RFC 6762 [8]. In such case, the first two non conflicting records are compared and the lexicographically later data wins. For instance, if two devices want to use the same local domain name (conflicting record) for two distinct IP addresses (non conflicting record), the device with the smaller IP address value must change its conflicting local domain name name. Once done, the device executes its initiating phase again and, hopefully, concludes it successfully. This is represented, in a simplified way, in Figure 3.

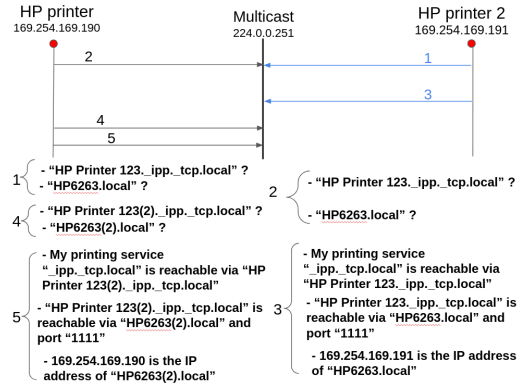

Fig. 3. Conflict in initiating phase

2) One device is in its initiating phase and the other in the standby phase:
In this case the device in the standby phase keeps its information unchanged while the other modifies its own to resolve the conflict. One such situation is represented in Figure 4 where "HP printer 2" is the one in standby phase.

3) Both devices are in the standby phase:
This can happen if, for instance, two parts of the network were initially disconnected and, then, reconnected. In such case, it could be that two distinct devices are in standby phase with conflicting attributes. A conflict will be detected if a third device queries either the local domain name or the local service instance(s) since two distinct devices will reply with the very same information. If this happens, the two devices MUST execute the initiating process and the resulting conflict will be treated as in the first case described here above.

III. State of the art

The so called *Man in the Middle (MITM)* attacks have been known for a while and exist in many different forms and flavours. We refer the interested reader to the survey by Conti et al. for a general introduction to the topic [21]. Our paper covers those specifically due to so called *Zeroconf protocols*.

These protocols have become popular in the last decade but had been in the making since 1999 [1]. Siddiqui et al. [22], in 2012, are the first ones to mention the existence of security issues related to these protocols. They briefly mention the possibility of MITM attacks and propose to run mDNS and DNS-SD on top of IPSEC for security reasons.

H. Raffiee, in 2015 [23], [24], produced an Internet draft that aimed at exploring the many different Zeroconf security issues. This is a well written document, quite interesting, but, unfortunately very brief. The explanations remain at a high level of abstractions. For instance, the report does not mention the notion of MITM attacks a single time.

In 2017, at the "Hack in the Box, HITB" conference, A. Atlasis made the first detailed presentation of several Zeroconf security issues [25]. There are no written proceedings but a video is available on the web which is quite instructive and covers topics such as the priority and weight in DNS SRV records, the cache-flush bit, etc.

In parallel, Bai et al., in 2016 and 2017 did publish two papers describing how to leverage flaws in the Bonjour pro-

tocol to run attacks against some application layer tools [12], [13]. They propose security countermeasures such as conflict detection and "speaking out" certificates. In a similar vein, one can mention the work published in 2019 by Stute et al. [14] in which the authors demonstrate the existence of vulnerabilities in the Apple Wireless Direct Link (AWDL) protocol. That protocol relies on mDNS and HTTPS [26] on top of Bluetooth Low Energy (BLE) or WiFi. The design flaws in mDNS are one of the elements that they leverage in order to carry out their attack.

In 2018, Erickson et al. introduce a tool dubbed Dreamcatcher in [27]. This tool aims at enforcing network access control. Among other things, it provides mechanisms to detect the arrival of new devices making mDNS announces.

From the list of past publications, one can draw a couple of important remarks:

1) It took more than 10 years to see the first publication on the design flaws of mDNS and 5 more years for a detailed public presentation.
2) Over the last few years, several applications have been shown to be vulnerable because they were relying on mDNS.
3) The mDNS attacks, in past publications, are seen as a simple mean to reach a more ambitious goal. Therefore, they are not explained in great details.

There are probably many more attacks that could be carried out because of the insecure foundation provided by mDNS. The growing IoT ecosystem is heavily relying on these protocols. This is why, in this paper, we explain as precisely as possible the many different ways a man in the middle attack can be carried out by misusing mDNS and DNS-SD. We highlight not only known design flaws but also reveal new problems due to poor implementation choices.

IV. Man in the middle attack

A. General Overview

To illustrate the notion of MITM attack, we will consider the simple case where a genuine user, Alice, is willing to print a file to a printer she knows to be available in the network. The attacker, Bob, tries to force Alice's machine to send the file to be printed to Bob's machine (without using ARP poisoning, another well known mechanism to carry out MITM attacks [28]–[30]). Afterwards, Bob sends the file, modified or not, to the printer in such a way that Alice does not notice that the file first went through Bob's machine before being printed where she wanted it to be. Throughout the text, we will use the following terminology to refer to Alice, Bob and the printer:

- Alice's machine is referred to as the client,
- Bob's machine is referred to as the attacker,
- The printer is referred to as the target of the attack.

As explained in Section II, the zero conf protocols rely on the bindings between three pairs of information:

1) A specific service of a device is associated with a local service instance name.
2) That local service instance is bound with, i.e., known to be reachable at, a local domain name.
3) That local domain name is bound with a specific IP address.

To launch a man in the middle attack (MITM), all an attacker needs to do is to break anyone of these three bindings. More precisely the MITM is possible through three technical attacks (bullets 1-3) and one social engineering attack (bullet 4), namely:

1) by "convincing" the clients that a specific local domain name is reachable at the attacker's IP address;
2) by "convincing" the clients that the local domain name associated with a requested local service instance is the one of the attacker;
3) by "convincing" the target of the MITM to change its local service instance;
4) by luring the end user to choose the wrong, similarly "looking", local service instance that he wants to use.

Furthermore, there are several parameters that one can play with when implementing any of these 4 attacks, leading to a large variety of ways to run them. The choices made by an attacker will enable him, or not, to maximize his chances of success and/or minimize the noticeable consequences of his attack.

In the following paragraphs, we present each of these four main attacks in some more details (Section IV-B) and, then, in Subsection IV-C we present different ways to run them. Next, in Subsection IV-D, we present their possible noticeable consequences and, at last, we present in Subsection IV-E the conditions that must be satisfied for some of these attacks to succeed. We have run a large number of experiments, trying all different forms of attacks against several targets. These experiments are described in Section V and all the results are summarized in a table, made available in [31], taking into account all the elements that we present here after.

B. Four attack strategies

1) Targeting the IP address associated to a local domain name: The first time that a client wants to print a file, she sends multicast DNS queries to find out which printer, if any, is available. Printers respond by providing in their replies, among other things, a so called *A record* that associates an IP address with their local domain name. If the attacker could replace the printer's IP address in this record with its own IP address, the attack would succeed. However, in the general case, it is impossible for an attacker to modify a packet sent by the printer to a multicast address. Instead, the attacker can send another packet, identical to that of the target but with a different IP address in the *A record*. If the client uses that information, the attack succeeds. This needs to be done carefully, by making the packet "invisible" to the printer to maximize the chances of success; if not, the printer would detect the conflict and react accordingly. The details on how to do that are covered later in Section IV-C. Figure

5 presents a simplified view of such possible exchange of packets that we can summarize as follows:

- Packet 1: The client looks for the available services.
- Packet 2: The devices announce the offered service(s).
- Packet 3: The client queries the printing services.
- Packet 4: Each printer provides its local service instance name, local domain name and IP address.
- Packet 5: The attacker sends to the unicast[1] address of the client the very same packet as the target but with a distinct IP address.

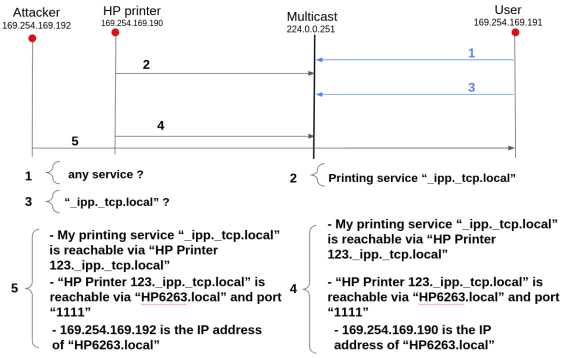

Fig. 5. Attack Strategy 1

Whether this attack will succeed or not depends on the minutiae of its modus operandi, as discussed in Section V.

We can detect this attack by looking for a local domain name associated to two distinct IP addresses. This could lead to false positives though, if, e.g., two identical devices are present in the network and have the same default local service and domain names.

2) Targeting the local domain name associated to a local service instance name: Instead of changing the IP address, the attacker could modify the local domain name associated to the target's local service name. This can be done by modifying the values of the *SRV record* linking these two, substituting the target's local domain name by the attacker's machine local domain name. Of course, the *A record* must also be changed to inform the client of the attacker's machine IP address. Figure 6 offers a simplified view of the packets exchanged for this attack to succeed[2]. The logic is the same as in the previous case, only the modified fields change. As in the previous case, the attacker must pay attention to the ways he runs this attack if he wants to maximize his chances of success (see Section V for more on this).

To detect this attack, we now have to look for a local service name associated to two distinct local domain names. As in the previous case, we could have false positives if two devices announce the same local service name but with a distinct local domain name.

[1]Replying to the multicast address could also work but with the risk of triggering a reaction by the printer if it detects the conflict.

[2]As in the previous case, the attacker replies to the unicast address to avoid any reaction from the target.

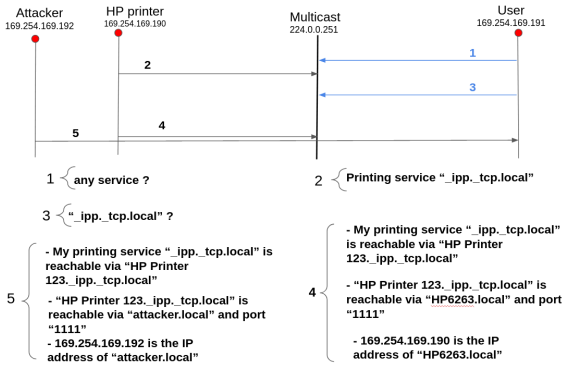

Fig. 6. Attack Strategy 2

3) Forcing the target to change its local service name: The last technical means to fool the client's machine is to force the target to change its local service name, effectively "stealing" it from it. This can be done by the attacker by announcing the same local service name as the target, intentionally triggering the conflict resolution process. Two possible scenari exist:

1) By carefully chosing the values of the non conflicting records, the attacker is certain to win the lexicographical order and the target will change its local service name, as explained in Section II-E, point 1 (page 3).
2) If, for whatever reason, the attacker cannot win with respect to the lexicographical order, there is still a way to force the target to change its local service name by acting in a non RFC compliant way. More precisely, if the target is in standby phase and if the attacker announces the same local service instance name, as explained in page 3, both machines MUST enter the conflict resolution phase, effectively moving back to the initiating state. If the attacker is not compliant with the RFC and remains in its standby phase, then it forces the other device, in the initiating phase, to change its name. In other words, by "standing its ground" (in a non RFC compliant behavior), the attacker forces the target to go back to the initiating phase while it remains in standby phase and, in that situation, uses the conflict resolution process to its advantage since it mandates to the only device in the initiating phase to resolve the conflict on its own. This is represented in a simplified way in Figure 7 where we can see that messages 4 and 5 make the conflict visible to anyone. We are clearly in the situation (3) of section II-E where both conflicting devices should revert to the initiating phase (situation (1) section II-E). As indicated by message 6, the client indeed engages into the conflict resolution phase by asking who, if anyone, is using his local service instance and,or, local domain name. With message 5, the attacker indicates ownership, in standby mode, of these values. From the target point of view, we are now,

effectively, in situation (2) (instead of (3)) of section II-E and the machine in the initiating phase, the target, must change its names. The problem could be solved if the RFC did explicitly specify what behavior to adopt when confronted with a non RFC compliant participant.

The net result of either of these techniques is that the attacker now has acquired the original legit local service name of the target and his local domain name. Any further use of that local service instance by any client in the network will lead them to contact the attacker instead of the target.

To modify the local service name in the packets, the attacker needs to modify several records, namely the *A record*, the *SRV record*, the *PTR record* and the *TXT record*. The roles of the first two have been mentioned before. The PTR record binds the local service name to the service and the TXT record provides additional information about the service (typically used by the application layer).

From a detection point of view, we will briefly observe a conflict with the same local service instance appearing in distinct packets. As explained before though, this could happen even in the absence of an attack. If the attack is carried out following the second method explained, the fact that one of the conflicting devices does not engage into the conflict resolution process can be seen as an additional red flag to consider.

A noticeable difference with respect to the two first attacks is that the target actively takes part to this third one by changing its attributes. As a result, all clients of the target are impacted. At the contrary, in the first two cases (when carried out thanks to unicast replies), the target was unaware of the ongoing attack and the sole impacted clients were the ones directly contacted by the attacker with a unicast reply.

4) Luring the user to choose the wrong service instance: A last way to proceed is to try to lure the end user instead of the machine itself, as represented in Figure 8. Instead of "stealing" the target's local service instance, Bob could advertise a name crafted in such a way that it could fool the end user. If, for instance, the various targets names are presented to the end user sorted by US ASCII character set value (ie alphabetical order, as seen by the user), the attacker can ensure that his chosen name will be at the top of the list if it starts with any of the 65 characters that exist before the first alphabetical letter, 'A' (hexadecimal value 41). By choosing, e.g., the hexadecimal value 01, an invisible character is in the first place and the local service instance becomes automatically the first in the list. Note that this is, somehow, user interface dependant because we have also seen cases where the name of the device, that appears in the user interface, is taken from the additional TXT field and, in such cases, the non printable characters could be stripped out during the parsing of the free form field.

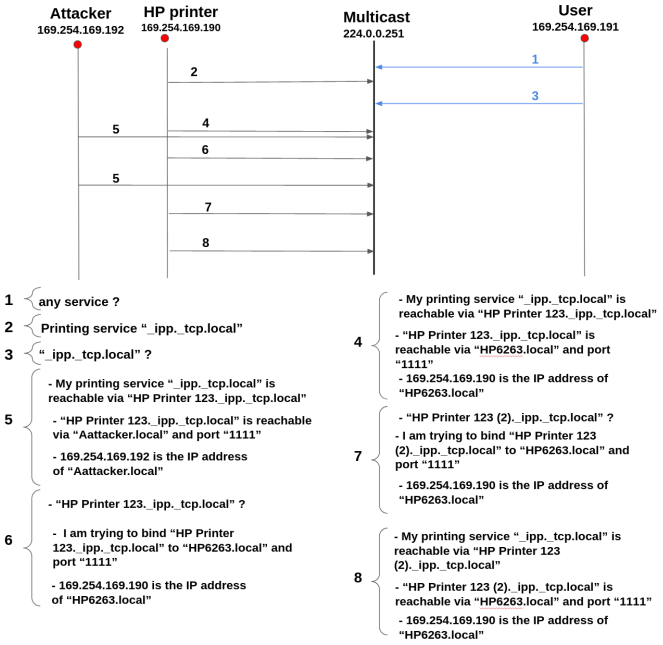

Fig. 7. Attack Strategy 3

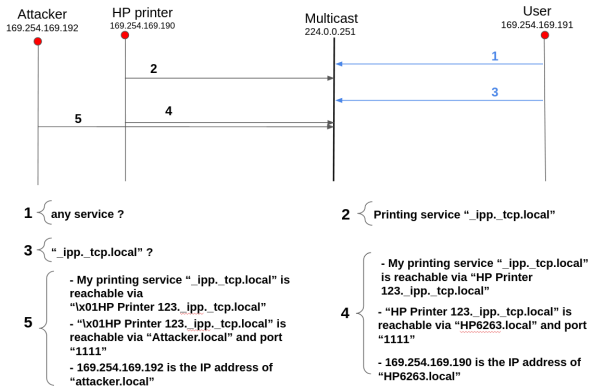

Fig. 8. Attack Strategy 4

C. Modus operandi

As eluded to, when describing the various attack strategies, there are a number of technicalities in the way they can be implemented that will influence not only the outcome of the attack but also the ability of a detector to observe abnormal situations. In this subsection, we dig deeper into the technical details to let the reader understand what are these various pecularities that we refer to. Here below, we describe six of them, namely, i) the cache flush bit, ii) the SRV record priority and weight, iii) unicast vs. multicast usage, iv) real vs. spoofed IP origin, v) sporadic vs. continuous attack, vi) timing of the attack.

1) Cache-flush bit: As explained in Section 10.2 of the RFC 6762 [8], a host has the possibility to flush the cache of all listening parties for a specific type of information. Quoting the RFC: " *When a resource record appears in a Resource Record Section of the DNS response with the*

cache-flush bit set, it means, "This is an assertion that this information is the truth and the whole truth, and anything you may have heard more than a second ago regarding records of this name/rrtype/rrclass is no longer true".

To maximise his chances of success, the attacker should use this bit whenever he sends information, to ensure that it will supersede the information previously sent by the target.

2) SRV record priority and weight: As seen before, The RFC 2872 defines the SRV DNS resource record to specify the location of services. The SRV RR also contains a weight and priority that clients must take into account to choose which of the, possibly, numerous servers to talk to when in need of a given service. More precisely, the RFC defines the priority and weight as follows:

Priority: A client MUST attempt to contact the target host with the lowest-numbered priority it can reach; target hosts with the same priority SHOULD be tried in an order defined by the weight field.

Weight: Larger weights SHOULD be given a proportionately higher probability of being selected.

The attacker, to maximise his chances to be chosen when several choices are possible for the recipient, should thus put a value of 0 for the priority and the highest possible value for the weight.

3) Unicast vs. Multicast: By default, all mDNS and DNS-SD communications are taking place using a multicast channel by sending all packets to the very same IP address 224.0.0.251. However, the authors of the RFC 6762 [8] have wisely identified a number of cases where it does make sense for a response to be sent to the unicast address of the querier. The RFC specifies that a " *... Multicast DNS querier MUST only accept unicast responses if they answer a recently sent query (e.g., sent within the last two seconds) that explicitly requested unicast responses. A Multicast DNS querier MUST silently ignore all other unicast responses [...] These questions requesting unicast responses are referred to as "QU " questions, to distinguish them from the more usual questions requesting multicast responses ("QM " questions)* ". Unfortunately, we have noticed during our experiments that some implementations are not compliant with that RFC and, therefore, make MITM easier to carry in a stealthy way. Indeed, we have found cases where a querier does **accept unicast responses even if it has sent a "QM " query**. Even worse, we have found cases where a machine does **accept a unicast response even if it has sent no query at all.**

From an attacker's point of view, the advantages of providing responses to the unicast address of the querier are obvious. Unicast replies will not be seen by the other devices and, in particular, not by the target, as eluded to in Figures 5 and 6.

4) IP Spoofing: Another convenient way to remain "invisible" from the target is to spoof its IP address when responding to the querier. Indeed, by default, a sender of datagrams to a multicast address will ignore its own packets. This is usually referred to as disabling the *ICMP_Multicast_Loopback*. Therefore, spoofing the target's IP achieves the same goal for the attacker than replying to the client's unicast address.

5) Sporadic vs. continuous: Once an attacker has succeeded in corrupting the information kept by a client, he needs to decide if this sporadic victory is enough or if he wants to ensure that the client's cache remains corrupted for the long run.

In the first case, the attacker has nothing to do. Each record is associated with a "Time to Live" value. Once that period expires the record is removed and the client obtains fresher, correct values, leaving no trace of the past attack.

At the contrary, to make his attack permanent, the attacker needs to periodically send fresh information, before the expiration period. If the attacker does that by using unicast or IP spoofing, he can remain undetected for a long time.

6) Timing of the attack: When running our experiments, we have noticed that the very same attack can lead to a very different outcome depending on the state the client is in. We have identified three distinct situations, explained here below:

1) **Idle** The client is connected to the network and has not issued any query. It stays idle. In that situation, if an attacker sends uninvited response packets, some client machines will happily accept them and update their local information with the content of the received packets. In that situation, we have also noticed that some target machines will not notice that a malicious user is trying to impersonate them even if the attacker does nothing to hide his packets. This is probably because they pay no attention to responses that do not correspond to any pending query.
2) **Refreshing** The information kept by a client has a certain 'time to live' (TTL). When that information expires, the client issues a request to refresh it. We have observed that some attacks can only be carried out successfully in such situation.
3) **Resetting** Service discovery and name resolution are services used by the applications such as, e.g., a printing or scanning application. We have seen that, on some machines, once these applications have obtained the information they needed they will store it and never ask for it again to their local stub resolver. As a result, our attacks can only have an effect on these applications if they are carried out before the application requests that information for the very first time. It is only if that device is deleted from the application or if the machine is rebooted that mDNS information will be fetched again by the applications and an attack made possible again.

D. Noticeable consequences

1) Multiple Services: As explained before, the result of some of these attacks will lead the end user to see the existence of two similarly looking services. For example, if the user search for a printing service he could end up discovering the two following services:

1) "HP [D2A9BE]._ipp._tcp.local";
2) "HP [D2A9BE](2)._ipp._tcp.local".

This is a simplified example. We have seen that it is possible to make things even more confusing for the user by having hidden characters inserted into one of the names, making the two services indistinguishable. However, one of these services is offered by the attacker whereas the other one is the legit one. If the user picks the wrong one, the attack succeeds and can be carried out in an invisible way for the client (i.e., the file is effectively printed on the expected printer). In the other case, it fails. With 2 services in the network, the attack has thus a 50% chance of success. Of course, the attacker could artificially increase the number of the malicious services to increase his chances of success. Such a tactic, however, is likely to catch the attention of the user who could start investigating why so many similarly looking services coexist.

2) Packet flooding: We have observed cases where failed attacks lead the target to generate an endless amount of messages. Whether the target service is reachable while this is ongoing is a function of the client and the target. The worst case scenario is a denial of service against the target (not a MITM).

The reason is that the attacker can force the target to repeat indefinitely the conflict resolution protocol. This is possible when a target, in standby phase, detects a conflict with the attacker. As per the RFC, it MUST enter conflict resolution phase. Two possibilities exist:

> **The attacker participates . . . :** . . . to the conflict resolution phase and "loses" the lexicographical order. At that point, the target reenters the standby phase. If the attacker does not change its name, a new conflict is detected that triggers the same sequence of events.
>
> **The attacker does not participate . . . :** . . . to the conflict resolution phase and, instead, remains silent up to the point where the target enters again the standby phase. At that time, the attacker emits its conflicting information again, triggering the same sequence of events.

We draw the attention of the reader to the subtle difference between these two situations and the one described in the third attack (case 2, on page 5). For that attack to succeed, the attacker must be announcing the conflicting information while the target is running the conflict resolution protocol. If the attacker is late, this leads to the situation exposed here above.

E. Success criteria

Based on our experiments, we have identified a certain amount of success criteria that must be met by the attacker for the MITM to succeed, we list the various situations below and detail them later in the text:

1) Race condition of last cache flush: In some cases, when the client receives several answers, the attacker wins only if its answer is the last one, with the cache-flush bit set to one, received by the client.

2) Race condition of first cache flush: In some cases, when the client receives several answers, the attacker wins only if its answer is the very first one, with the cache-flush bit set to one, received by the client. The reason is that, after having received the first information, the client ignores the next ones until resetting the device.

3) Establishing TCP Connection: In some cases, it is not enough to craft the right mDNS packets to fool a client. The attack will only succeed if, afterwards, the client can successfully establish a TCP connection with the application it is expected to talk to. The attacker must be able to accept such connection and be able to "speak" the correct application layer protocol[3].

4) Selecting the right choice: In the social engineering attacks, since we add a new machine or a new service instance to the user interface, our attack will only really succeed if we manage to fool the user to select the fake entry.

V. EXPERIMENTS

A. Testbed definition

In theory, the possibilities of running a man in the middle attack can be derived from the analysis of the protocols specifications. In practice, it is well known that implementations may drift away from an RFC. Also, there are often some corner cases that are ill specified or not specified at all. This is the reason why we have built a small testbed made of two distinct clients and two distinct targets. This is definitely not enough to enable us to generalize our results but it is sufficient to already demonstrate the diversity of the observed behaviours. More precisely, the testbed is defined as follows:

- The attacking code runs on a Kali Linux 5.8.0 machine.
- The first client machine runs on a Ubuntu 18.04 LTS machine. The second one runs on Windows 10.
- The first target was a HP OfficeJet Pro 6230 printer. The second one was an Apple TV (3rd gen model A1469) running Apple TV software 7.2.2.
- All these devices were connected thanks to a Cisco Linksys router E900. The attacker is connected using the WiFi while the others are connected with Ethernet Cables.
- On the Ubuntu client, we have used

[3]In our case, we forward these TCP connections to the target thanks to iptables SNAT forwarding rules [32].

  - the default built in user interface to manage printers;
  - Avahi [11] to discover existing services in our network;
  - Airstream [33], a third party software, to stream a video to the Apple TV.
- On the windows client, we have used
  - the built-in User interface to manage printers;
  - Apple Bonjour for Windows [34] to discover existing services in our network;
  - Soda player [35], a third party software to stream a video to the Apple TV.
- We are making all the attack scripts available on github. [31]
- The same github repository contains a pdf file containing the results of the attacking scripts we have used during the experiments.

B. Methodology

We have carried out the four attack strategies described in Section IV-B using the various modus operandi described in Section IV-C. For the first three technical attacks, this led to a grand total of 288 attacks while the fourth one did only generate 12 distinct attacks because several variations were irrelevant in that case.

All the technical attacks have been carried out with the cache-flush bit set to 1, the SRV priority value set to 0 and the weight value to 100 (we have never observed a legit device using such a high value).

The table in pdf format available in the github repository details the results obtained for all the 300 cases. In the Appendix, we offer a summarized version of it in which we have grouped together all attacks that led to the same output, keeping the technical and social engineering separated. The meaning of each column is defined as follows:

A: Index value of the attack;
B: Client machine type, 2 possible values: Linux (1) or Windows (2);
C: Target device, 2 possibles values: HP printer (1) or Apple TV (2);
D: Attack type, 4 possible values corresponding to the four cases from Section IV-B (page 4): IP (1), domain name (2), service instance (3), social engineering (4);
E: destination IP used by the attacker, 2 possible values: multicast IP (1) or client's unicast IP (2);
F: Source IP used by the attacker, two possible values: the real attacker's IP (1) or the target's spoofed IP (2);
G: value 1 indicates that the attacker runs its attack only once; value 2 indicates that the attacker keeps sending reminders regularly (to ensure persistence of the attack by refreshing the poisoned information);
H: The timing of the attack, with respect to the state of the client, as defined in point 6 of Section IV-C (page 6); three possible values: idle (1), refreshing (2), resetting (3);
I: the required criteria for the attack to succeed, as defined in Section IV-E (page 8); five possible values: winning the race condition for the last cache flush (1), winning the race condition for the first cache flush (2), being able to establish a TCP connection (3), having the user select the "right" choice (4), none (0);
J: The outcome of the attack is encoded with 8 different values and several of these codes can appear together for a given attack; success without noticeable consequence (1); a possible continuous flood of packets (2); numerous unwanted responses sent at regular intervals (3); the OS acknowledges the existence of 2 devices for a given service (4); user interface shows two similarly looking service instances (5); refresh requests from the user for the erroneous records (6); failure (7); possible success conditional to a success criterion (typically winning a race condition) (8).

Due to the lack of space, we cannot go through all the various cases but, instead, we prefer to offer a synthetic discussion of the results obtained for the 288 technical attacks and to share some lessons learned. The interested reader is referred to [31] for the detailed observations.

Among the 288 technical attacks:

184 attacks did succeed.
104 attacks did fail systematically.
104 of the successful attacks were only so when a success criterion was met, which was not systematic (race condition problem).
30 attacks, even when successfull, led to a flood of packets easily noticeable.
16 attacks, to remain successful, needed the attacker to send frequent reminders.
56 attacks left a noticeable fingerprint for the user, typically at the application user interface.

Among other things, these experiments highlighted the following noticeable elements, mentioned earlier:

- Linux machines accept unicast responses when they should not, even without having sent a request. This makes them very vulnerable to a number of attack scenarios.
- On windows machines, as soon as an application has used the information at his disposal to establish a TCP connection to the device, it can not be attacked anymore, until a reset.
- Target devices do not detect that they are impersonated if they have not first replied to a query.
- Target devices do accept to abort the conflict resolution process when the other conflicting party stands

its ground (i.e., they are RFC compliant but do not notice that the other device is not).
- Even in networks configured thanks to a DHCP and DNS server, machines rely on mDNS to contact local machines, making these attacks possible despite the administrators having "properly" configured their networks.

VI. Detection methods

Due to the severity, simplicity and large applicability of these attacks, we thought that it was responsible to provide some detection technique against them. We have thus written a couple of Zeek (aka Bro) scripts (around 450 loc). They are very experimental and by no means optimized. We encourage others to revisit and improve them. These scripts are grouped into 4 sets. The first one detects suspicious events, the last three look at specific elements of a given attack method. We have run these scripts to detect successfully our various attacks and had them running in a home office environment for a couple of weeks without generating false positives others than the ones mentioned earlier in the text. However, we acknowledge that this does not constitute a thorough and sound evaluation of their quality. We are in the process of deploying them in larger environments to accumulate more data on the risks of false positives. Regarding false negatives, all the attacks, as implemented, have been successfully detected by our scripts. Again, this does not ensure that an astute attacker could not run them in a way that could defeat our detection.

A. Detecting unexpected elements

Every attack scenario relies on fake mDNS responses. This is thus what we try to detect. In particular, our scripts look for the following things:

- a new unknown device in the network; this will generate a false positive when introducing a new legit device.
- Two mDNS responses having the same Domain name or the same service name; this will generate a false positive, in particular, for devices that do not introduce some amount of randomness in their names.
- Two services with similar services names: for example: "HP [D2A9BE]._ipp._tcp.local." and "HP [D2A9BE](2)._ipp._tcp.local."
- Periodic mDNS responses without queries.
- Unicast addresses used in mDNS replies.

B. Detecting floods of packets

The packet flooding is easy to perceive. Two devices keep sending queries and responses continuously which Zeek can easily recognize.

C. Detecting the race condition for the first cache flush

In order to win the race condition, the attacker has two possibilities:

- Listen to the query and send a response. This is the normal behaviour of the devices. But, it has 50% chance of success depending on which answer arrives first. If the target's answer reaches the client first, the attack fails. In this case we detect the attack by checking if there are 2 responses for the same service.
- Packet flooding: when a new client joins the network, the attacker proactively starts sending replies with very short delays to impersonate a target. That could give him a better, yet small, chance to win the race when the client will send a query. This situation is similar to the packet flooding and detecting the attacker presence is easy in this case.

D. Detecting the race condition for the last cache flush

To achieve the attack in this case, the attacker will modify every answer sent by the legitimate device and sends it to the client after a short delay. We can detect it by observing the traffic.

VII. Ethical Discussion

Some of our attacks originate from the fact that the RFC 6762 [8] does not specify what behavior genuine participants to the protocol should adopt when confronted with non compliant ones. Whether this is a vulnerability of that RFC or an issue orthogonal to it, is open for debate. In any case, we have exchanged with S. Cheshire, one of the authors of the RFC, who has suggested to request agenda time to present this at the DNSSD working group session at the upcoming IETF meeting. He also points out that application layer end to end encryption is available for a certain amount of devices, such as Airprint printers, which could, if well configured, mitigate these problems.

Regarding Avahi not being compliant with respect to unicast replies, we have had several fruitful and constructive exchanges with Trent Lloyd, the person leading this open source project. He acknowledged the problem and is working on a fix. He also pointed out that Avahi implements all the checks preconised (SHOULD not MUST) in the RFC 6762 on the TTL value and source IP address. This severly limits the impact of the problem to the sole machines located within the same LAN.

We make our code available to facilitate the reproduction of results by researchers but its design and coding style make it ill suited and fragile for someone to run real attacks with it in a non controlled environment.

VIII. Conclusions

Our 300 experiments have enabled us to demonstrate the feasibility and the severity of MITM attacks enabled by the zeroconf protocols. They have not only revealed how non RFC compliant participants could run attacks but also that non compliant implementation render these attacks easier and stealthier. It is our hope that this work will be an eye opener for those who only see the usability benefits of these protocols and miss the threats they represent.

References

[1] Zero configuration networking (zeroconf) working group homepage. [Online]. Available: www.zeroconf.org

[2] S. Cheshire, "private communication," 2021.

[3] S. Cheshire and M. Krochmal, "Requirements for a protocol to replace the appletalk name binding protocol (nbp)," RFC 6760, February, Tech. Rep., 2013.

[4] P. Black and J. Melanson, *Inside Macintosh: Networking*. Addison-Wesley Publishing Company, 1994.

[5] S. Cheshire. (1997) Nbp/ip (name binding protocol over ip). [Online]. Available: http://www.stuartcheshire.org/rants/NBPIP.html

[6] S. Cheshire, B. Aboba, and E. Guttman, "Dynamic configuration of ipv4 link-local addresses," Internet Requests for Comments, RFC Editor, RFC 3927, May 2005.

[7] B. Aboba, D. Thaler, and L. Esibov, "Link-local multicast name resolution (llmnr)," Internet Requests for Comments, RFC Editor, RFC 4795, January 2007.

[8] S. Cheshire and M. Krochmal, "Multicast dns," Internet Requests for Comments, RFC Editor, RFC 6762, February 2013, http://www.rfc-editor.org/rfc/rfc6762.txt. [Online]. Available: http://www.rfc-editor.org/rfc/rfc6762.txt

[9] ——, "Dns-based service discovery," Internet Requests for Comments, RFC Editor, RFC 6763, February 2013, http://www.rfc-editor.org/rfc/rfc6763.txt. [Online]. Available: http://www.rfc-editor.org/rfc/rfc6763.txt

[10] Apple bonjour. [Online]. Available: https://developer.apple.com/bonjour/

[11] Avahi. [Online]. Available: https://avahi.org/

[12] X. Bai, L. Xing, N. Zhang, X. Wang, X. Liao, T. Li, and S.-M. Hu, "Staying secure and unprepared: Understanding and mitigating the security risks of apple zeroconf," in *2016 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2016, pp. 655–674.

[13] ——, "Apple zeroconf holes: How hackers can steal iphone photos," *IEEE Security & Privacy*, vol. 15, no. 2, pp. 42–49, 2017.

[14] M. Stute, S. Narain, A. Mariotto, A. Heinrich, D. Kreitschmann, G. Noubir, and M. Hollick, "A billion open interfaces for eve and mallory: Mitm, dos, and tracking attacks on ios and macos through apple wireless direct link," in *28th {USENIX} Security Symposium ({USENIX} Security 19)*, 2019, pp. 37–54.

[15] V. Paxson, "Bro: A system for detecting network intruders in real-time," *Computer networks*, vol. 31, no. 23-24, pp. 2435–2463, 1999.

[16] S. Haas, R. Sommer, and M. Fischer, "Zeek-osquery: Host-network correlation for advanced monitoring and intrusion detection," in *IFIP International Conference on ICT Systems Security and Privacy Protection*. Springer, 2020, pp. 248–262.

[17] Corelight company home page. [Online]. Available: https://www.corelight.com/

[18] D. E. 3rd and A. Panitz, "Reserved Top Level DNS Names," Internet Requests for Comments, RFC Editor, BCP 32, June 1999.

[19] R. Droms, "Dynamic host configuration protocol," Internet Requests for Comments, RFC Editor, RFC 2131, March 1997, http://www.rfc-editor.org/rfc/rfc2131.txt. [Online]. Available: http://www.rfc-editor.org/rfc/rfc2131.txt

[20] D. C. Plummer, "Ethernet address resolution protocol: Or converting network protocol addresses to 48.bit ethernet address for transmission on ethernet hardware," Internet Requests for Comments, RFC Editor, STD 37, November 1982, http://www.rfc-editor.org/rfc/rfc826.txt. [Online]. Available: http://www.rfc-editor.org/rfc/rfc826.txt

[21] M. Conti, N. Dragoni, and V. Lesyk, "A survey of man in the middle attacks," *IEEE Communications Surveys Tutorials*, vol. 18, no. 3, pp. 2027–2051, 2016.

[22] F. Siddiqui, S. Zeadally, T. Kacem, and S. Fowler, "Zero configuration networking: Implementation, performance, and security," *Computers & electrical engineering*, vol. 38, no. 5, pp. 1129–1145, 2012.

[23] H. Rafiee, "Multicast dns (mdns) threat model and security consideration," *Internet Eng. Task Force*, vol. 102014, 2015.

[24] ——, "Multicast dns (mdns) threat model and security consideration," Working Draft, IETF Secretariat, Internet-Draft draft-rafiee-dnssd-mdns-threatmodel-03, May 2015, http://www.ietf.org/internet-drafts/draft-rafiee-dnssd-mdns-threatmodel-03.txt. [Online]. Available: http://www.ietf.org/internet-drafts/draft-rafiee-dnssd-mdns-threatmodel-03.txt

[25] A. Atlasis, "An Attack-in-Depth Analysis of multicast DNS and DNS Service Discovery," slides presented in 2017, Tech. Rep. [Online]. Available: https://conference.hitb.org/hitbsecconf2017ams/materials/D2T2-Antonios Atlasis-An-Attack-in-Depth Analysis of Multicast DNS and DNS Service Discovery.pdf

[26] E. Rescorla, "HTTP Over TLS," Internet Requests for Comments, RFC Editor, RFC 2818, May 2000, http://www.rfc-editor.org/rfc/rfc2818.txt. [Online]. Available: http://www.rfc-editor.org/rfc/rfc2818.txt

[27] J. Erickson, Q. A. Chen, X. Yu, E. Lin, R. Levy, and Z. M. Mao, "No one in the middle: Enabling network access control via transparent attribution," in *Proceedings of the 2018 on Asia Conference on Computer and Communications Security*, 2018, pp. 651–658.

[28] C. L. Abad and R. I. Bonilla, "An analysis on the schemes for detecting and preventing arp cache poisoning attacks," in *27th International Conference on Distributed Computing Systems Workshops (ICDCSW'07)*. IEEE, 2007, pp. 60–60.

[29] D. Bruschi, A. Ornaghi, and E. Rosti, "S-arp: a secure address resolution protocol," in *19th Annual Computer Security Applications Conference, 2003. Proceedings.* IEEE, 2003, pp. 66–74.

[30] S. Y. Nam, D. Kim, and J. Kim, "Enhanced arp: preventing arp poisoning-based man-in-the-middle attacks," *IEEE communications letters*, vol. 14, no. 2, pp. 187–189, 2010.

[31] Github repository for attack code and zeek scripts. [Online]. Available: https://github.com/dh001996/MITM_MDNS/tree/main

[32] O. Andreasson *et al.*, "Iptables tutorial 1.2. 2," *Copyright© 2001–2006 Oskar Andreasson, GNU Free Documentation License*, 2001.

[33] Airstream. [Online]. Available: https://linux.softpedia.com/get/Multimedia/Video/AirStream-102979.shtml

[34] Apple bonjour for windows. [Online]. Available: https://support.apple.com/kb/DL999?locale=en_US

[35] Soda player. [Online]. Available: https://www.sodaplayer.com/

Appendix

Synthetic presentation of the experimental results

The complete table with the detailed results for each experiment is available in [31]. We have included in the next page a synthetic version of the results in which we have merged together all lines that were having the same outcome. When different values were present in a given column for a set of merged lines, we have indicated the distinct values separated by "or". The reader is invited to look at the full table to see which exact combination of columns values leads to the outcome indicated in the J column.

A	B	C	D	E	F	G	H	I	J
Attack Index	Client Type	Target	Attack type	Source IP	Destination IP	Duration	Timing	Success Criteria	Outcome
									success=1
			IP address = 1					RC last=1	flood=2
	Linux = 1	HP printer=1	domain name =2	attacker IP=1	Multicast = 1	sporadic=1	Idle = 1	RC first=2	unwanted responses=3
	Windows=2	Apple TV=2	service instance=3	Spoofed IP =2	Unicast =2	continuous=2	Refresh=2	TCP = 3	2 devices=4
			social eng.=4				Reset=3	select=4	2 services=5
								nothing=0	erroneous refresh=6
									failure=7
									success if criteria met=8
1 - 32	1	1 or 2	1 or 2	1 or 2	1 or 2	1 or 2	1	0	1
33-48	1	1 or 2	1 or 2	1 or 2	1 or 2	2	2	0	3
49-72	1 or 2	2	3	1 or 2	1 or 2	1 or 2	1 or 2	4	5
73-80	1	2	3	1 or 2	1 or 2	1 or 2	2	4	6
81-184	1 or 2	1 or 2	1 or 2 or 3	1 or 2	1 or 2	1 or 2	1 or 2 or 3	0 or 1 or (2 and 3)	7
185-227	1 or 2	1 or 2	1 or 2	1 or 2	1 or 2	1 or 2	1 or 2	1 or (2 and 3)	8
228-256	1 or 2	1 or 2	1 or 2	1 or 2	1 or 2	1 or 2	2 or 3	1 or (2 and 3)	2 and 8
257-263	1	1	2	1 or 2	1 or 2	1 or 2	3	1	4 and 8
264-287	1 or 2	1 or 2	3	1 or 2	1 or 2	1 or 2	3	(1 and 4) or 4	5 and 8
288	1	1	2	1	1	1	3	1	2 and 4 and 8
289, 292	1	1 or 2	4	-	-	-	1	0	5
290, 293	1	1 or 2	4	-	-	-	2	0 or 3	6
295-297	2	1	4	-	-	-	1 or 2 or 3	0	7
298-299	2	2	4	-	-	-	1 or 2	0	5 and 7
291,294,300	1 or 2	1 or 2	4	-	-	-	3	4	5 and 8