HTB Sherlock: Noxious

forensics ctf hackthebox htb-sherlock dfir sherlock-noxious wireshark llmnr llmnr-poisoning pcap capinfos net-ntlmv2 hashcat

Sep 4, 2024

In part three of HackTheBox's beginner-focused active directory Sherlock series, I'll look at a PCAP showing an LLMNR poisoning attack. I'll see the attack based on a typo in the hostname of an SMB share the victim is trying to visit. I'll show how the victim connects to that share on the attacker, and how the attack can crack the victims password from there.

Challenge Info

Name	Noxious Play on HackTheBox
Release Date	1 August 2024
Retire Date	1 August 2024
Difficulty	**Very Easy**
Category	SOC
Creator	CyberJunkie Moderator 2 613 hackthebox.com

Background

Scenario

> *The IDS device alerted us to a possible rogue device in the internal Active Directory network. The Intrusion Detection System also indicated signs of LLMNR traffic, which is unusual. It is suspected that an LLMNR poisoning attack occurred. The LLMNR traffic was directed towards Forela-WKstn002, which has the IP address 172.17.79.136. A limited packet capture from the surrounding time is provided to you, our Network Forensics expert. Since this occurred in the Active Directory VLAN, it is suggested that we perform network threat hunting with the Active Directory attack vector in mind, specifically focusing on LLMNR poisoning.*

Notes from the scenario:

- I'm looking for LLMNR traffic.
- I'm given a network capture (PCAP) file.
- This is an active directory environment, and I should be looking for LLMNR poisoning.

Questions

To solve this challenge, I'll need to answer the following 9 questions:

1. Its suspected by the security team that there was a rogue device in Forela's internal network running responder tool to perform an LLMNR Poisoning attack. Please find the malicious IP Address of the machine.
2. What is the hostname of the rogue machine?

3. Now we need to confirm whether the attacker captured the user's hash and it is crackable!! What is the username whose hash was captured?
4. In NTLM traffic we can see that the victim credentials were relayed multiple times to the attacker's machine. When were the hashes captured the First time?
5. What was the typo made by the victim when navigating to the file share that caused his credentials to be leaked?
6. To get the actual credentials of the victim user we need to stitch together multiple values from the ntlm negotiation packets. What is the NTLM server challenge value?
7. Now doing something similar find the NTProofStr value.
8. To test the password complexity, try recovering the password from the information found from packet capture. This is a crucial step as this way we can find whether the attacker was able to crack this and how quickly.
9. Just to get more context surrounding the incident, what is the actual file share that the victim was trying to navigate to?

Data

The download contains a single PCAP file:

```
oxdf@hacky$ unzip -l noxious.zip
Archive:  noxious.zip
  Length      Date    Time    Name
---------  ---------- -----   ----
137211904  2024-06-24 16:44   capture.pcap
---------                     -------
137211904                     1 file
```

Tools

For packetcapture data, I'll be using Wireshark. `capinfos` is a nice tool to quickly get information about the PCAP, such as the timeframe. It comes with the standard Wireshark installation.

Background

Link-Local Multicast Name Resolution (LLMNR) is a protocol used to resolve hostnames to IP addresses without needing DNS. It operates on local networks, allowing devices to discover each other by broadcasting requests to all devices on the same network segment. When a device wants to connect to another by name, it sends an LLMNR query asking if any device on the network has the requested name. The device with the matching name responds with its IP address, enabling the connection. LLMNR is used in small networks that don't have a dedicated DNS server, or as a fallback mechanism in Windows active directory environments don't get an answer from DNS.

The security risk of LLMNR is that it's suscepitble to poisoning, where an attacker is able to respond before the legitimate host, or when there is no legitimate host with the searched name. In either case, the victim then tries to connect to the attacker, where the authentication can be captured and cracked offline using bruteforce.

Results

PCAP Overview

Timeframe

I'll use `capinfos` to get stats about the PCAP (and use `TZ-UTC` to get times in UTC):

```
oxdf@hacky$ TZ=UTC capinfos -a -e capture.pcap
capinfos: An error occurred after reading 29303 packets from "capture.pcap".
capinfos: The file "capture.pcap" appears to have been cut short in the middle of a
packet.
  (will continue anyway, checksums might be incorrect)
File name:           capture.pcap
First packet time:   2024-06-24 11:17:22.462145
Last packet time:    2024-06-24 11:40:07.259807
```

The capture takes place over a 13 minute timeframe on June 24, 2024.

PCAP Statistics

On opening the PCAP in Wireshark, at the bottom right it shows that it has a lot of data:

A good way to get a quick handle on what kind of data is present is under Statistics –> Protocol Hierarchy, which shows how the data breaks down:

I'll notice there's LLMNR as expected, as well as SMB, HTTP, SSH, Kerberos, RPC, and ARP.

The Statistics –> Endpoints tab is also nice for getting bearings. I like to shot different ways, and by "Port" column is often helpful:

Right away I'll note that 172.17.79.0/24 seems to be the local network. SSH is open on .130. .4 seems to be a domain controller (DC) with Kerberos (88) and RPC (135). There's a lot of outbound HTTPS (443) traffic. Scrolling down, there's another probably Windows host on .136 with SMB open, as well as SMB on the potential DC as expected:

Someone has an RDP session into .136. The traffic outbound seems to be mostly from .129, .130, .135, and .136.

Domain Controller

Before moving focus to LLMNR, I need to understand more about what the network looks like, especially verifying where the DC is. From statistics, I suspect it is 172.17.79.4. I'll filter on that host (`ip.addr==172.17.79.4`). There are multiple exchanges that confirm my suspicion.

For example, packet 10077 is a NetBIOS host announcement broadcast traffic (because it's going to the subnet's broadcast address, .255) showing the hostname DC01:

Packets 9281-9288 are .136 authenticating to .4 over Kerberos (which fails, but it's still acting like a DC):

.136 tries to load the share `\\DC01\IPC$` and `\DC01\DC-Confidential on .4 over SMB starting at packet 10179:

At this point it's fair to say the DC is .4.

LLMNR Poisoning

Identify Attack

Given the prompt, I'll start with the LLMNR, which is as simple as entering "llmnr" into the display filter bar. Now there are only 126 packets:

In this data, there are many queries for DCC01. If this were DC01, I'd expect to see .4 responding with it's IP, but instead it's 172.17.79.135 responding with it's own IP (Task 1):

DCC01 is the typo the user entered trying to reach DC01 (Task 5). The queries are coming from 172.17.79.136. It's worth noting also the IPv6 addresses for both the attacker (fe80::2068:fe84:5fc8:efb7) and the victim (fe80::7994:1860:711:c243).

The attack happens from 11:18:30-11:34:46.

First Attack

Just after the first instance of LLMNR poisoning, the victim connects to the attacker on SMB over IPv6:

It starts the SMB connection, then tries to get a Kerberos authentication, which fails. Looking at the TGS-REQ, that's because it's requesting a ticket for the CIFS service (SMB) on DCC01, which doesn't exist:

Then it falls back to NTLM auth, where it tries to authenticate as john.deacon (Task 3) in the FORELA domain. This is all happening at 11:18:30 (Task 4)

Reassembling the Damage

Theory

To see what was exposed, I'll look at rebuilding the NetNTLMv2 challenge response that the attacker could have attempted to crack. This is often referred to as a hash, though it's not actually a hash in this case. It's a challenge sent by the server to the client that the client encrypts with the NTLM hash of the user and sends back. The server uses it's copy of that hash to decrypt and make sure it's correct.

An attacker who sees the challenge data and the encrypted response can brute force passwords to see if they can decrypt the data, and if so, they have the user's password.

Collect Parts

This post from 801 labs gives a nice recap on how to build a hash that can be used with `hashcat` from a PCAP. I'll need the:

- username - already have john.deacon
- domain - already have FORELA
- server challenge
- NTproofstring
- modified NTLMv2 response

In packet 9291, the server sends the client the challenge, "601019d191f054f1" (Task 6):

The next packet from the client has the NTproofstring and the response:

The NTproofstring is "c0cc803a6d9fb5a9082253a04dbd4cd4" (Task 7), and I'll need to remove that from the front of the NTLMv2 Response to get "010100000000000080e4d59406c6d…[snip]…00000000".

All of this comes together to make the "hash":

```
oxdf@hacky$ cat john.deacon.hash
john.deacon::FORELA:601019d191f054f1:c0cc803a6d9fb5a9082253a04dbd4cd4:01010000000000008
```

Hashcat

I'll pass this to `hashcat` with `rockyou.txt` to see if it cracks. `hashcat` auto-detect mode will identify the hash if I've built it correctly:

```
$ hashcat john.deacon.hash /opt/SecLists/Passwords/Leaked-Databases/rockyou.txt
hashcat (v6.2.6) starting in autodetect mode
...[snip]...
Hash-mode was not specified with -m. Attempting to auto-detect hash mode.
The following mode was auto-detected as the only one matching your input hash:

5600 | NetNTLMv2 | Network Protocol
...[snip]...
JOHN.DEACON::FORELA:601019d191f054f1:c0cc803a6d9fb5a9082253a04dbd4cd4:01010000000000008
...[snip]...
Started: Sun Sep  1 15:08:03 2024
Stopped: Sun Sep  1 15:08:07 2024
```

It took four seconds to get the password "NotMyPassword0k?" (Task 8).

Other Enumeration

Post Attack

After the attack, the victim on .136 does eventually make a connection to the DC on SMB:

The share path is `\\DC01\DC-Confidential` (Task 9).

Later in the PCAP, there's a remote desktop connection from the attack into the victim's machine as john.deacon:

This is a good indication that the attack was successful at compromising the account.

Attacker Info

Looking a bit more at what else the attacker did, I'll filter on their IPs (`ip.addr==172.17.79.135 or ipv6.addr==fe80::2068:fe84:5fc8:efb7`). Right at the top of the resulting traffic is a DHCP Discover packet:

At the end it offers a hostname of V17VT3M03. Unfortunately, that's no accepted as the correct answer. If I look for other DHCP traffic from this host, there's a request at packet 12714:

This one shows the hostname kali (Task 2).

Question Answers

1. Its suspected by the security team that there was a rogue device in Forela's internal network running responder tool to perform an LLMNR Poisoning attack. Please find the malicious IP Address of the machine.

   172.17.79.135

2. What is the hostname of the rogue machine?

   kali

3. Now we need to confirm whether the attacker captured the user's hash and it is crackable!! What is the username whose hash was captured?

   john.deacon

4. In NTLM traffic we can see that the victim credentials were relayed multiple times to the attacker's machine. When were the hashes captured the First time?

   2024-06-24 11:18:30

5. What was the typo made by the victim when navigating to the file share that caused his credentials to be leaked?

   DCC01

6. To get the actual credentials of the victim user we need to stitch together multiple values from the ntlm negotiation packets. What is the NTLM server challenge value?

   601019d191f054f1

7. Now doing something similar find the NTProofStr value.

   c0cc803a6d9fb5a9082253a04dbd4cd4

8. To test the password complexity, try recovering the password from the information found from packet capture. This is a crucial step as this way we can find whether the attacker was able to crack this and how quickly.

   NotMyPassword0k?

9. Just to get more context surrounding the incident, what is the actual file share that the victim was trying to navigate to?

   `\\DC01\DC-Confidential`