Feature, not bug: DNSAdmin to DC compromise in one line

Shay Ber · Follow
7 min read · May 7, 2017

Background

In addition to implementing their own DNS server, Microsoft has also implemented their own management protocol for that server, to allow for easy management and integration with Active Directory domains. By default, domain controllers are also DNS servers; DNS servers need to be reachable and usable by mostly every domain user. This, in turn, exposes quite some attack surface on domain controllers — on one part, the DNS protocol itself and on the other, the management protocol, which is based on RPC.

We will shallowly delve into the protocol's implementation and detail a cute feature (certainly not a bug!) which allows us, under some circumstances, to run code as SYSTEM on domain controllers, without being a domain admin. Although this is certainly not a security vulnerability (so no panic is needed), as confirmed with Microsoft, it's still a cute trick which can be useful as an AD privilege escalation in red team engagements.

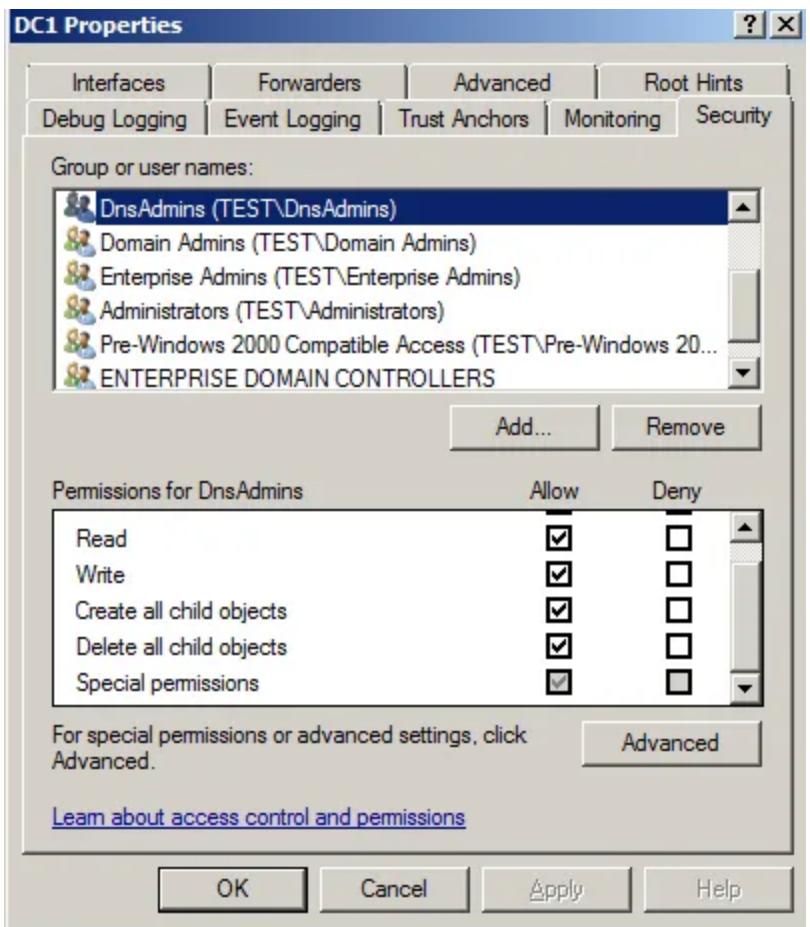
All presented information was gathered by reading the protocol specification ([MS-DNSP], https://msdn.microsoft.com/en-us/library/cc448821.aspx) and reverse engineering the dns.exe binary using IDA.

DNS Server Management Protocol basics

The management protocol layers on top of RPC, which, for this protocol's purposes, can be layered on top of either TCP or named pipes. If you're interested in messing around with the protocol or its implementation, it can be found in domain controllers under c:\windows\system32\dns.exe. Its RPC interface UUID is 50ABC2A4–574D-40B3–9D66-EE4FD5FBA076 and it uses the \PIPE\DNSSERVER named pipe for transport.

The DNS server runs as a service on domain controllers. The management interface can be accessed by running dnsmgmt.msc and connecting to an AD DNS server (usually a domain controller). It allows users to configure,

among other things, DNS zones, lookups, caching, forwarding and logging. Several objects in this 'hierarchy' are securable — DNS server objects (which are *not* computer accounts), zone objects and records. In this case, we're interested in server objects, whose ACL on a fresh install should look something like this:



default ACL for DNS server object

By default only DnsAdmins, Domain Admins, Enterprise Admins, Administrators and ENTERPRISE DOMAIN CONTROLLERS have write access on this object. Notably, from an attacker's perspective, if we're members of each group but DnsAdmins, we've already owned the domain. So, let us see what we can do if we've got ourselves a DnsAdmin.

. . .

Hunting PDFs

This is where the protocol specification comes to our aid. Section 3.1.4: Message Processing Events and Sequencing Rules, basically details all operations which the server needs to support. The first one is R_DnssrvOperation, which contains a pszOperation parameter, which determines the operation performed by the server. While scrolling over the huge list of possible pszOperation values, we see this:

ServerLevelPluginDll	On input **dwTypeId** MUST be set to DNSSRV_TYPEID_LPWSTR, and **pData** MUST point to a Unicode string that contains an absolute pathname for server side plug-in binary on the DNS server or an empty Unicode string.

Right, we can tell the server to just load a DLL of our choice! Awesome! Upon searching the spec for ServerLevelPluginDll, we find the following useful piece of information:

- If pszOperation is ServerLevelPluginDll, the server MUST store the value passed in pData and return SUCCESS, indicating only that the value was successfully received. The server MUST NOT validate the value passed, nor attempt to load the DLL, until the server is restarted. When the server restarts, if the value stored for ServerLevelPluginDll is not an empty string, the server MUST attempt to load the DLL specified. If the DLL fails to load for any reason, the server MUST fail to start. If the DLL has been loaded, then whenever the server is required to invoke the DLL query function, the server MUST invoke the query function of the DLL with a query name and type and add any resulting records to the server's cache. Whenever the server processes a query, if the DLL has been loaded, the server MUST invoke the query function of the DLL in the following conditions:
  - If a query cannot be answered with the information already present in the server's zone database and cache, then prior to recursing (if applicable), invoke the DLL query function and try again to answer the query from local data.
  - If a response SHOULD have records in its additional section but no such records are in the server's cache or zone database, invoke the DLL query function and attempt again to find records for the additional section.

Looks like the server does not even do any verification on the dll path specified in this operation. Before starting to implement this, I figured someone must have dug this up before. A google search for ServerLevelPluginDll raised nothing of the sort, however it did pop up the useful dnscmd command line tool, which I hadn't known before.

Luckily, dnscmd already implements everything we need. A quick look at its help message and another glance at https://docs.microsoft.com/en-us/windows-server/administration/windows-commands/dnscmd give us the following option:

dnscmd.exe /config /serverlevelplugindll \\path\to\dll

First, trying to run this as a weak domain user with no special permissions on the DNS server object (other than Generic Read, which is granted to all members of the Pre-Windows 2000 Compatible Access group, which by default contains the Domain Users group), the command fails with an access denied message. If we give our weak user write access to the server object, the command no longer fails. This means that members of DnsAdmins can successfully run this command.

Still lazy enough not to use IDA, trying to run this on a domain computer running with a member of DnsAdmins while running process monitor and process explorer on our DC, we see that no DLL is loaded into dns.exe's address space, as expected. We do see, however, that the following registry key is now populated with the path we sent:

HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\services\DNS\Parameters\ServerLevelPluginDll.

Great. Now, for testing purposes, we restart the DNS server service. Whoops — it fails to start, and clearing the registry key value allows it to start. Apparently it needs something more from our DLL. Time to open IDA.

There are several possibilities to quickly reach the functionality we seek to reverse in this case — searching for relevant strings and searching for relevant APIs are usually the easiest and quickest. In our case, going through all xrefs to LoadLibraryW or GetProcAddress gives us what we need — going through the code of the function which LoadLibraryW's our DLL and the one function it is called from, we see that no validation is performed at all on the path given to ServerLevelPluginDll.

The hurdle we've run into is indeed the only one — if either the DLL fails to load, or it does not contain one of the DnsPluginInitialize, DnsPluginCleanup or DnsPluginQuery exports, the service will fail to start. We also need to make sure that our exports all return 0 (success return value), otherwise they may cause the service to fail as well.

The pseudocode for the function responsible for loading the DLL is roughly as follows:

```
HMODULE hLib;
if (g_pluginPath && *g_pluginPath) {
  hLib = LoadLibraryW(g_pluginPath);
  g_hndPlugin = hLib;
  if (!hLib) {...log and return error...}

  g_dllDnsPluginInitialize = GetProcAddress(hLib,
"DnsPluginInitialize");
  if (!g_dllDnsPluginInitialize) {...log and return error...}
  g_dllDnsPluginQuery = GetProcAddress(hLib, "DnsPluginQuery")
  if (!g_dllDnsPluginQuery) {...log and return error...}
  g_dllDnsPluginCleanup = GetProcAddress(hLib, "DnsPluginCleanup")
  if (!g_dllDnsPluginCleanup) {...log and return error...}

  if (g_dllDnsPluginInitialize){
    g_dllDnsPluginInitialize(pCallback1, pCallback2);
  }
}
```

Here's a quick PoC to demonstrate how the code for such a DLL should look under Visual Studio 2015:

Sample code for our plugin dll

The pragmas are in place to modify the default export names to the ones that we desire. To verify that our exports are good, we can use dumpbin /exports path\to\dll.

Now we try running dnscmd with our new dll and voila, it works! All we need is to put our dll on a network path which is accessible by one of the domain controllers' computer accounts (dns.exe runs under SYSTEM) (Read access for the Everyone SID should do the job), and we can run code as SYSTEM on a domain controller, thus taking control of the domain.

While this demonstrates that it is possible to take over a domain if you're a member of DnsAdmins, it's not limited to just that — all we need to successfully pull off this trick is an account with write access to a DNS server object. The ACLs for these objects are usually, from my experience, not kept as clean or monitored as ACLs for domain admins (or similar groups protected by AdminSDHolder), thus offering a nice chance for a small domain elevation of privilege.

As also noted in the spec, this should work across all recent Windows Server versions:

ServerLevelPluginDll support across OS versions

Microsoft's MSRC have been contacted regarding this issue and have stated that it will be fixed by basically only allowing DC administrators to change the ServerLevelPluginDll registry key, and that it will be possible to toggle this feature off in future releases.

Regardless, dns.exe currently still runs as SYSTEM and boasts a good amount of attack surface, so it is probably a worthwhile candidate for some fuzzing — both for the DNS implementation and the management interface.

. . .

Disclosure timeline

Mar. 31st — Initial disclosure to secure@microsoft.com.

Apr. 1st — Disclosure acknowledged and forwarded for review.

Apr. 8th — MSRC case 38121 opened.

May 2nd — Investigation concluded and ruled that this is not a vulnerability and will be fixed in the future, outside a security update.

May 5th — Discussions regarding further hardening of the DNS server service.

May 8th — Findings are published.

Big thanks to MSRC's Daniel for the handling of this case, was a pleasure to work with.

All in all, reading Microsoft's protocol specifications can be a useful pasttime, even more so when you search for 'DLL' inside them :)

Happy hunting!

. . .

Small update (May 10th): Nikhil Mittal has elaborated on the precise technicalities of getting this feature to work in his lab, here. Thank you!

DNS Red Team Microsoft Active Directory

 --   

Written by Shay Ber

95 Followers

Follow