Waiting for goDoH

sensepost.com/blog/2018/waiting-for-godoh/

Reading time ~12 min

Posted by Leon Jacobs on 24 October 2018
Categories: Bypass, Command execution, Dns, Experiment, Ioc, Malware, Research, Tools, Tunnelling

or DNS exfiltration over DNS over HTTPS (DoH) with godoh

"Exfiltration Over Alternate Protocol" techniques such as using the Domain Name System as a covert communication channel for data exfiltration is not a new concept. We've used the technique for many years at SensePost, including Haroon & Marco's 2007 BH/DC talk on Squeeza. In the present age this is a well understood topic, at least amongst Infosec folks, with a large number of resources, available, online that aim to enlighten those that may not be familiar with the concept. There are also practical techniques for detecting DNS Tunnelling on your network.

Using DNS as a covert communication channel has many benefits when considering the monitoring capabilities of a target. By utilising a protocol that underpins technologies such as email and web browsing, one may not immediately expect DNS to be used for anything other than, well, DNS. Alas, it is possible to use a completely legitimate protocol for two-way communication in and out of a network, abusing a possibly overlooked monitoring opportunity (if not a necessity). This technique does not come without cost to the attacker though. DNS covert communication is one of the slowest methods of the several options an attacker may have. Especially when compared to malware that make use of HTTP/S to communicate.

Many organisations have also managed to adjust their architectures and monitoring to defend against this two decade old idea, since malware and attackers alike have taken advantage of it. Using split horizon DNS, monitoring the size and rate of requests as well as analysing the labels in a lookup all provide opportunities to detect and prevent successful tunnelling. From an attackers perspective, this increases complexity when trying to stay under the radar while still having a reliable channel back into the network. By tweaking the behaviour of the DNS-channel such as the length of a hostname's labels and the rate of requests detection can be bypassed but this will almost always come with a speed trade off. Simply not allowing recursive name lookups to the outside world would obviously also prevent the attack, but at the cost of some usability. Thankfully for defenders, monitoring

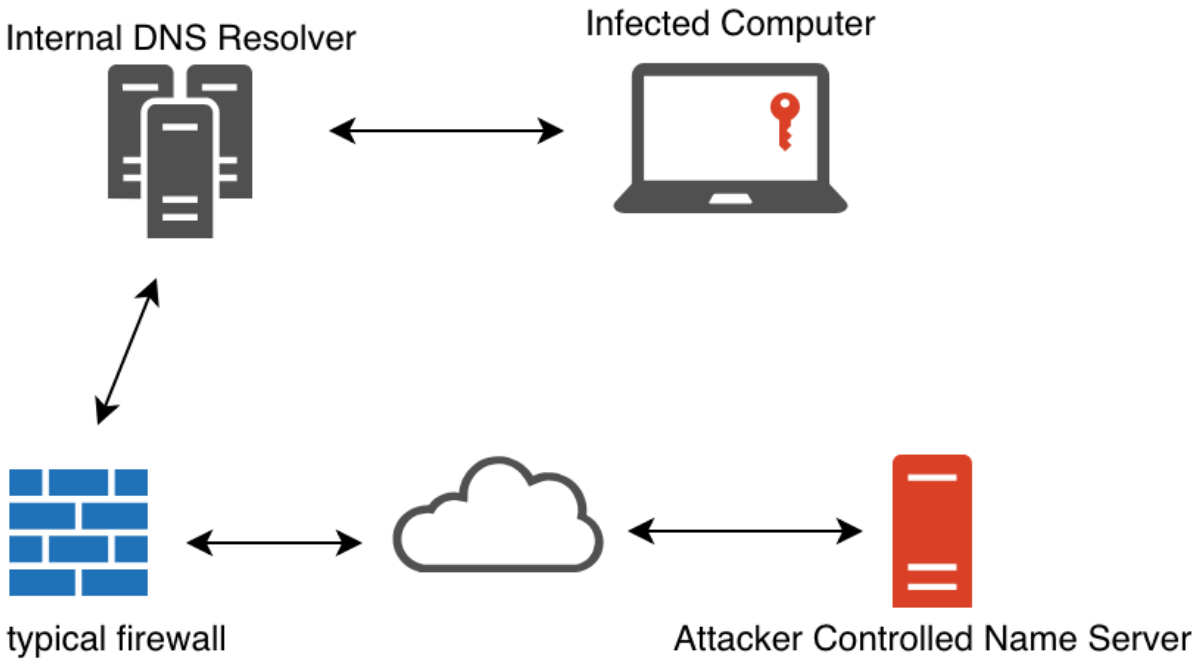
DNS is relatively simple in that you only need to focus on DNS at a protocol level, and probably only from the few caching forwarders you allow to make recursive, outbound lookups to the world.

Until now. Fast forward a few years from the original DNS tunnelling discussions, and we are presented with a new RFC that describes a protocol called DNS-over-HTTPS, or DoH. Meet RFC8484. Basically, it is possible to have a full RFC1035 compliant DNS conversation, over HTTPS. Think of it as a JSON API to make DNS lookups. A simple HTTP GET request and a predictable JSON response format, all via a provider such as Google. Right there is where attacker spidey senses should have tingled. Using legitimate and more often than not, trusted domains such as google.com to "front" your traffic to a C2 has been a thing for a while. Domain Fronting has seen a decent enough uptake where it has been used for censorship circumvention as well as in some malware campaigns. The present struggles of domain fronting aside, by abusing DNS over HTTPS we can once again achieve the same level of evasion, albeit at a much slower rate but using a **trusted** domain.

I built a quick proof of concept called "godoh" with the purpose of demonstrating DoH as an exfiltration channel, but also to give defenders some tooling to test with so that monitoring and detections for this technique can be built. I figured this DoH exfiltration technique was a neat idea, but literally while writing this post I have since discovered other existing mentions of this technique. And yesterday at the recent ATT&CK conference, @dtmsecurity also released some tooling for red teamers to make use of this exact same idea in the popular adversary simulation toolkit, Cobalt Strike. Nonetheless, allow me to take you through my thought process and finally give you some tooling to play with this on your own networks.

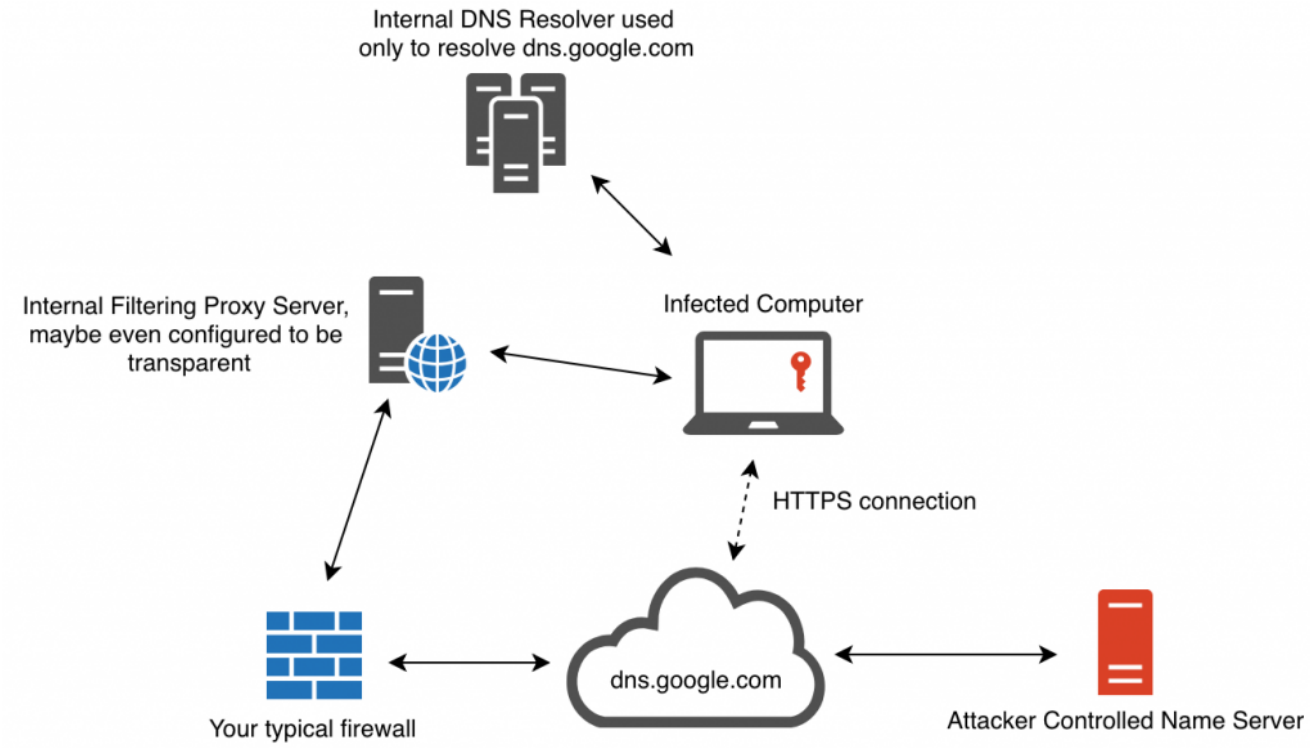
Imagine you are on a network that has relatively good DNS monitoring. They are capable of detecting its use as an exfiltration channel (or don't allow external zones to resolve via DNS in the first place). They also have a fancy Layer7 proxy/firewall that does URL content classification and strictly blocks based on that. The google.com domain is whitelisted for whatever reason, and therefore https://dns.google.com/ probably isn't blocked.

A simple piece of malware making use of DNS tunnelling may typically have code running on a computer that periodically polls for commands to run, and responds with the output of those commands encoded as a series of A record lookups to an attacker controlled domain that are reconstructed server-side.



Example traffic flow for traditional C2 using DNS.

I ported the exact same protocol idea to simply use a DNS over HTTPS provider for exactly the same lookups. The lookups themselves did not change (and don't need to, we are still talking valid DNS remember), but merely the transport used to ask DNS questions and parse an answer.



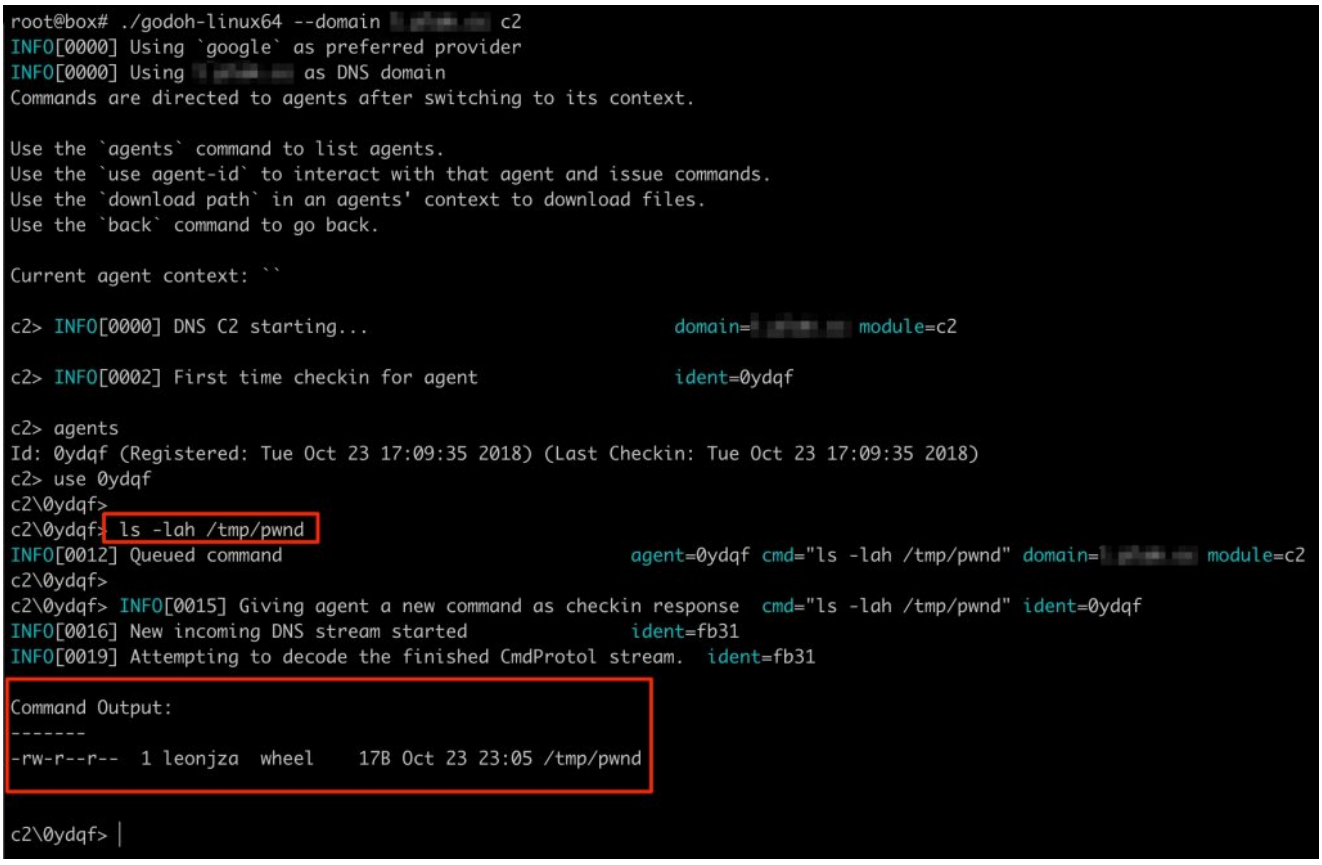
Modified traffic flow, fronting DNS lookups using a provider such as Google.

Consider this architecture and the monitoring implications for a moment. Now, one can no longer rely on the fact that DNS is a very specific protocol that could be sinkholed and controlled on a network, but instead, we know have the added complexity of an HTTPS

connection, **to an often trusted domain such as google.com**, proxying these DNS requests in and out of a network. Once a request is made to a DoH provider such as https://dns.google.com/, they in turn use traditional DNS to resolve a name and respond with an answer to the DoH client. But it's not just Google, there are many DoH providers available today, and one can quickly see this is a harder problem to solve all of a sudden.

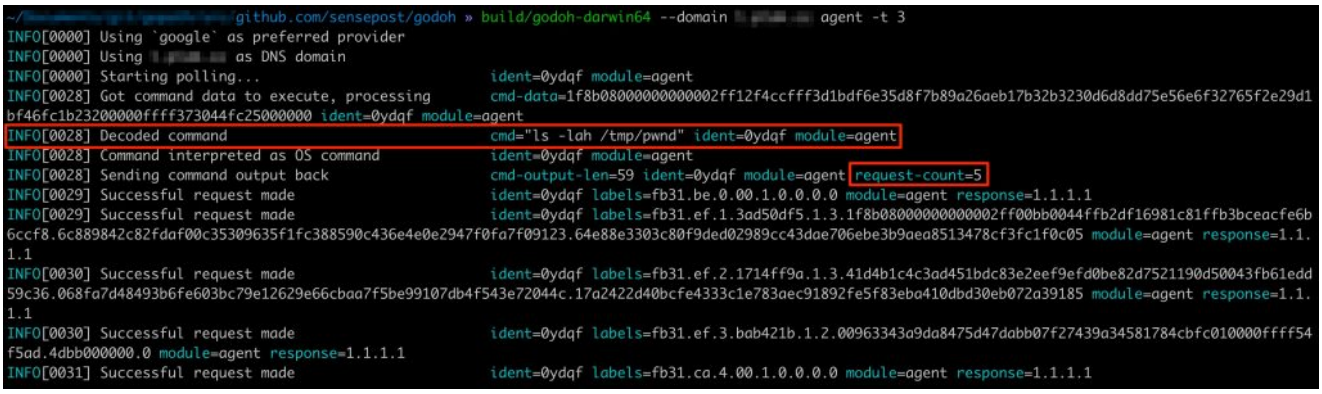
Since `godoh` is written in Golang, a single executable for most platforms can be built that contains both the server-side and client-side code needed. Just like traditional DNS tunnelling, you would need to configure a domain (or subdomain) to have its name server point to your c2 server and run the `godoh c2` command from there. The `c2` command starts a DNS server specifically geared towards understanding how to have conversations with agents using DNS. On the client side, the `godoh agent` command is run to connect to a c2 using any of the DoH providers supported by godoh. It is here where the agent and a DoH provider have conversations that are finally translated into questions for the c2 DNS server to answer. Simple, yet effective.

In practice, the server-side component for `godoh` looks like this:



godoh server-side C2 interactions example.

In the screenshot above, a new agent connected to the C2 and the command `ls -lah /tmp/pwnd` was issued to be executed by the agent. Once completed, the agent sent the output back to the C2 (in the form of a series of DNS A record lookups) where the server finally decrypted the payload and presented the output to the screen. On the agent side, the invocation and execution of the command looked as follows:



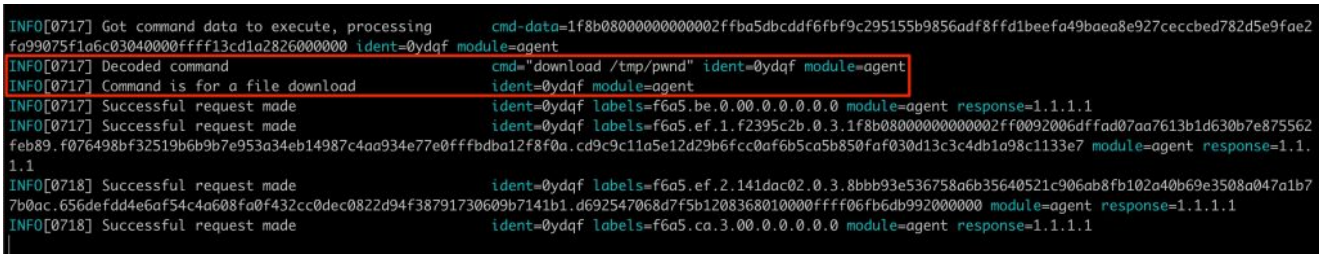
godoh agent interaction

As you can see, the agent started to poll for new commands (every three seconds as indicated by the `-t` flag) via DNS TXT record lookups. Once a command was received, it was decrypted and executed on the host. Once done, the output was sent back to the C2 server via DNS A record lookups for a total of 5 requests for the complete conversation. What is interesting to note here is that the client knows if the server successfully received a packet (and could decrypt and validate a crc32 checksum) based on the IP address in the response data. A response of `1.1.1.1` indicates a successful receipt and decryption of data.

It is also possible to download files with `godoh`. Keep in mind that this is still DNS, which is limited in packet size, so downloading large files take **lots** of requests, which directly translate into **lots** of time.

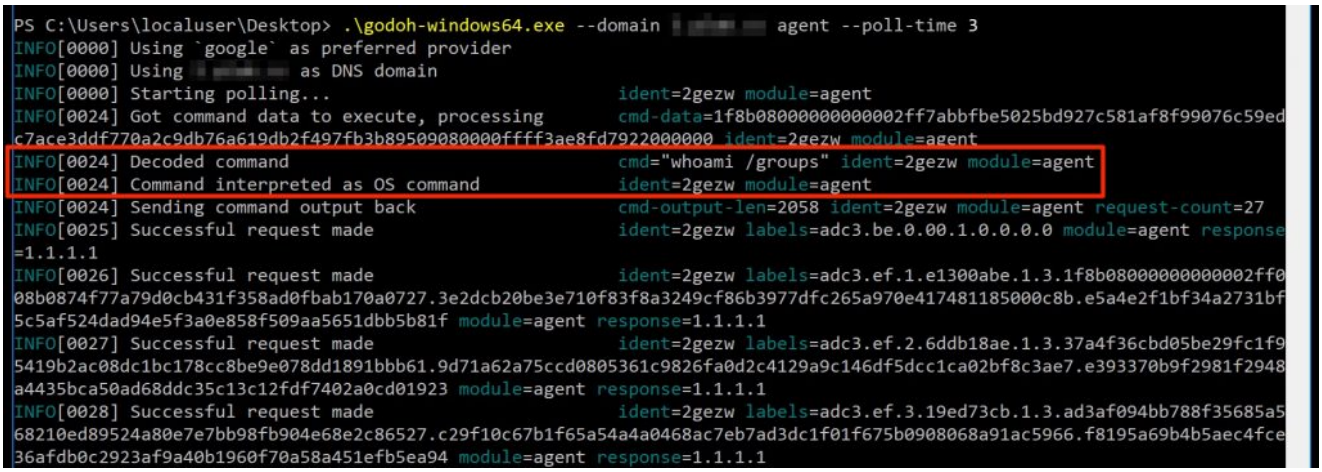
godoh file download example.

By simply issuing the `download /tmp/pwnd` command, the agent read the contents of the target file and sent it back to the C2.



godoh agent file download

From the agents perspective, downloading a file simply means reading its contents and sending it back to the C2 with DNS A record lookups all using DoH.

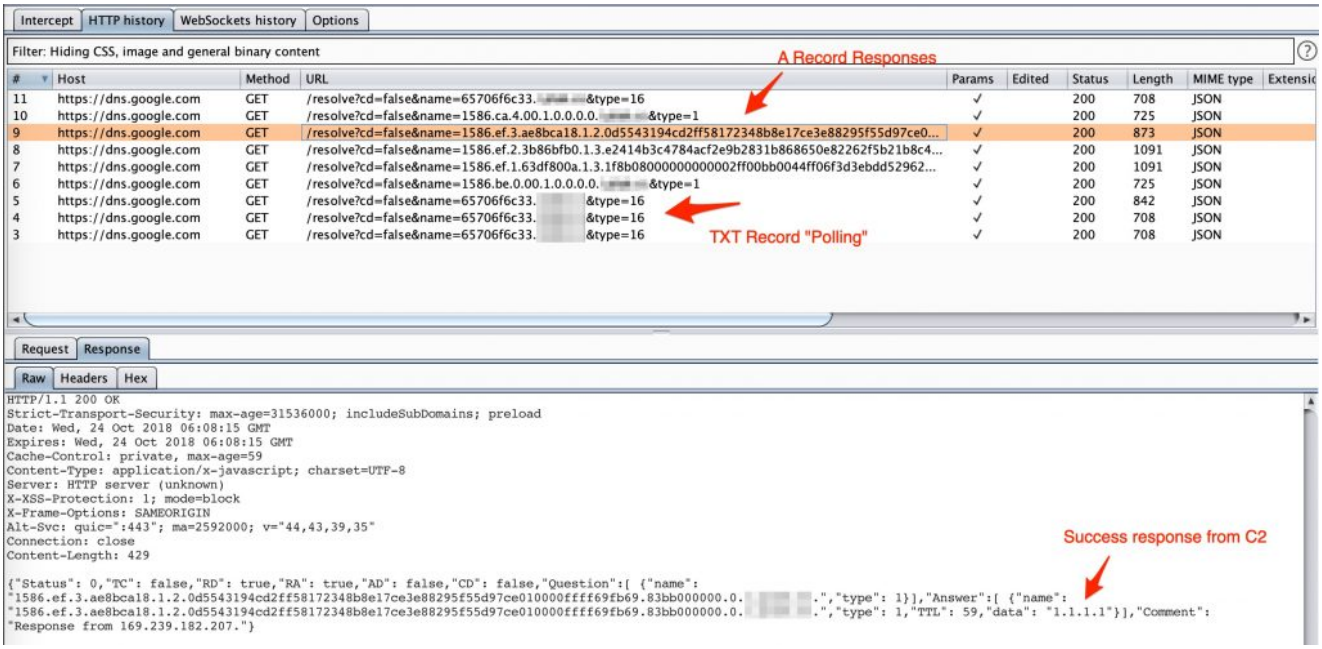


godoh Windows client executing the whoami /groups command and returning the output to the C2.

What you should have noticed in the screenshots for the agent output by now is the contents of the labels fields. These are the full hostnames for the specially crafted protocol that would have been appended to the target domain to form DNS lookups. If you are interested in the details, this code comment attempts to shed some light on the meanings of the labels themselves. The server-side component interpreting data can be found here. In short, the protocol works as follows:

- Once started, make a DNS TXT record lookup to the target domain in the form of `agentidentifier.targetdomain`.
- Once the server receives the lookup, if it is the first time seeing this agent identifier the C2 would record the existence of this new identifier as a potential target to interact with.
- If there are no commands to be executed, just respond with the default "no commands" response. This loop repeats itself infinitely.
- If there **are** commands to be executed, encode, compress and encrypt the command and send it along within the TXT lookup response.
- The agent then parses the TXT record response and decrypts, decompresses and decodes the command to execute it.
- When complete, the output goes through a process of encoding and encryption, and finally translation , where the encrypted data is translated into chunks to be sent as DNS A record lookups.
- Server side, a control flag is read to understand if an incoming request starts a new stream, is part of a pending stream or is the end of a stream and decides based on that what the next step should be.
- Once a DNS lookup stream is complete, the protocol type is checked (as in, is this simply command output to be echoed to screen or a file download where the contents should be saved to file) and the appropriate action is taken.

When looking at this traffic using an HTTP proxy, this conversation looks something like this:



godoh C2 communications example viewed in a HTTP proxy.

If you are keen to play with this in your own environment then you can get `godoh` here: https://github.com/sensepost/goDoH. Prebuilt binaries are available, but keep in mind that they are built using a publicly known encryption key available in the source code. Ideally, you should build your own versions with a unique key, easily generated with `make key` before `make`.