March 2023
Geoff Huston

Hiding behind MASQUEs

It has been almost a decade since Edward Snowden exposed a program of mass surveillance by the US NSA, using the Internet for large scale data harvesting. The Internet had been profligate in the way in which various protocol scattered user data around with a somewhat cavalier disregard for privacy. This profligacy was largely unconscious at the time as far as I can tell. This was the way these protocols had always worked, and while we had taken the lessons about hostile parties and adverse attacks seriously by then we were not as sensitive to the issues of the way in which user data was gathered and used in unconstrained ways. We had not really thought about the relationship between application transactions and the common network infrastructure as being one that could be potentially a hostile one.

The IETF responded to these Snowden disclosures with RFC7258, "Pervasive Monitoring is an Attack", published in May 2014. The document outlined an IETF intention to mitigate pervasive monitoring through deliberate actions in protocol design that were intended to make it far harder to undertake pervasive monitoring. This call for better protection of application and user data in network protocols was not an isolated cry of anguish from the geeks at the IETF! This perspective resonated with many parts of the content and application industry, who had similar concerns, probably for slightly different reasons. Advertising plays a large role in funding the provision of these services, and the advertising programs are based on knowledge of the user. Allowing this data about users and their usage to leak in unconstrained ways exposes one of the critical assets of any advertising platform to potential competitors. Taking measures to occlude various network transactions from external inspection resonated strongly which this content sector.

We've seen a number of changes over the past decade as a result of this call for action. The most notable change has been the shift of the default web browser and server behaviour to use encrypted communications via Transport Layer Security (TLS). This measure protects the content of the network transaction from third party onlookers, as the keys used to encrypt the web session are known only to the two end parties and communicated between the two parties in a secure mode. However, a casual observer for these HTTPS transactions can still glean useful metadata from observing the packet exchange. The client-side IP address is still visible to the observer in the IP packet header, so the client's identity can still be inferred (to some extent). The domain name part of the URL of the server-side of the web transaction also visible in clear text in the packet exchange that is used in the TLS initialisation handshake as the Server Name Indication (SNI) field in the TLS initialisation exchange. (Encrypting this last part of the TLS handshake is still a work-in-progress for the TLS Working Group of the IETF.) And, of course the time of day is a common constant. In other words, despite TLS session encryption, some aspects of a web transaction are still visible to an onlooker, namely the *who*, *when* and *where* of a transaction. What is obscured with the use of TLS is the detail of the web transaction, or the *what* component of the transaction.

There have been a number of efforts to counter this form of exposed metadata, and one interesting approach is described in the DNS protocol, using an approach called *Oblivious DNS over HTTPS* (ODoH), specified in RFC9230. This approach uses two trusted network intermediaries in sequence, a *Proxy* and a *Target*, both selected by the user. A "bundle" is created, consisting of the DNS query and a response encryption key. This bundle is encrypted using the Target's public key. The client then initiates a DNS

over HTTPS (DoH) session with a Proxy and passes to the Proxy the IP address of the Target and the encrypted bundle. The Proxy then initiates a DOH session with the Target and passes the encrypted bundle as a DNS query to the Target. Only the Target is able to decrypt the bundle using its private key and access the original DNS query and the response key. The Target can then resolve the query using conventional DNS query mechanisms. The Target then encrypts the response into a bundle using the response key and passes it back to the Proxy over the Proxy/Target DoH session. The Proxy then relays this bundled response to the original client over the Client/Proxy DoH session. The client can decode the bundle and retrieve the DNS response (Figure 1).

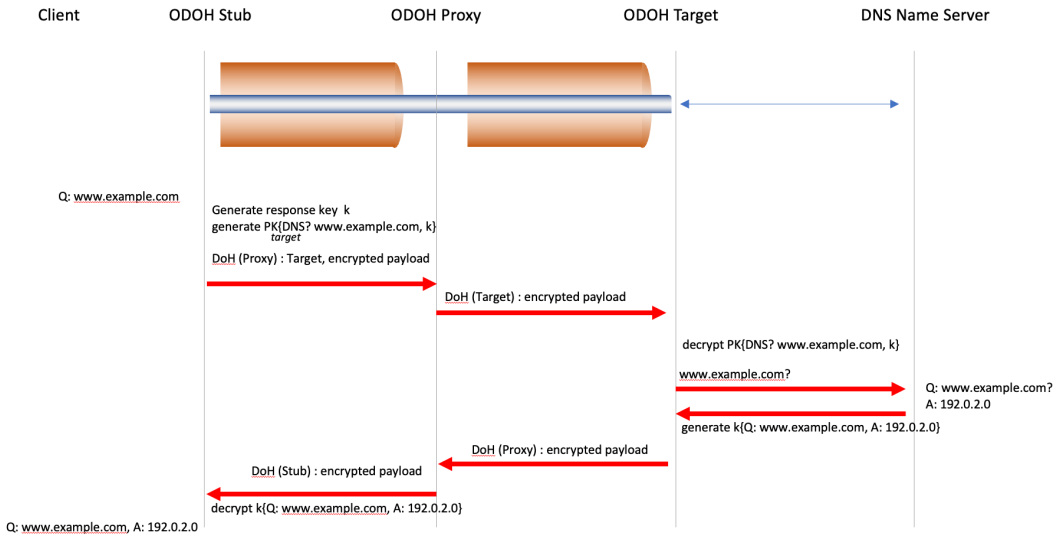

Figure 1 - Oblivious DNS Framework

In Oblivious DNS the Proxy is aware of the IP address of the client, but not the DNS name that is the subject of the client's query. The Target is aware of the DNS name being queried, but not the client's identity.

The next question is: Can this ODoH approach be extended to work in other HTTP contexts? This is the question that the Oblivious HTTPS (OHTTP)effort is working on. The way that OHTTP attempts to address this is to deconstruct the transaction across multiple parties in pretty much the same manner as ODoH. Aside from a nomenclature change the overall framework is very similar, which is not surprising considering that both the DNS and HTTP are basically simple transaction (command/response) protocols.

The OHTTP approach is to use a similar form of dual layer encryption framework, where the client encrypts the URL request with the public key used by the Gateway. This encrypted message is then packaged in a relay request which is sent to the Relay. The Relay receives the encapsulated request directly from the Client, along with the standard HTTP request headers and related network connection information as a part of the HTTPS session between the client and the Relay. The Relay uses a new HTTPS session to Gateway to send a Gateway request, with the body of the request being the encapsulated request from the client. The Relay does not know what data the Client has sent to the Target.

The Gateway processes the gateway request, as the inner encapsulated request was encrypted using the Gateway's public key it can decrypt the request. The Gateway opens an HTTPS session with the target web server, using the information provided by the client in the client's original URL request. Responses from the target are encapsulated and encrypted and passed back along the Gateway's HTTPS session with the Relay. The relay then takes the encapsulated response and passes it back to the client, who can then decrypt the response (Figure 2).

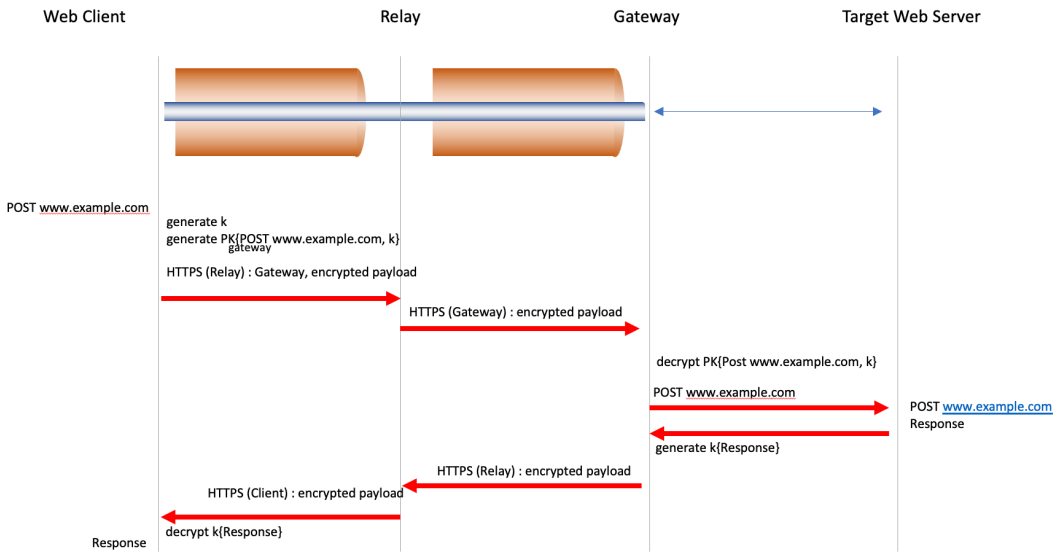

Figure 2 - Oblivious HTTPS Framework

The client is aware of its own identity, and the URL to whom the request has been made. The Relay is aware of the IP address of the client, and the identity of the Gateway to whom the request will be passed, but not the identity of the Target, nor the URL that is to be passed to the Target. The Gateway is aware of the identity of the Target and the URL, but only knows the identity of the Relay, and not the client. And the Target its aware of the URL and the identity of the Gateway, but not the identity of the Relay or the end client. If the Relay and the Gateway are able to combine their information, then they could reconstruct the identity of the client and the URL that the client is accessing, so it's an essential attribute of this framework that the Relay and Gateway are mutually independent.

The relationship between the client and the Gateway is a bigger issue, in that the client is implicitly trusting the Gateway to act with integrity. There is no end-to-end relationship between the client and the Target at this stage, and the client is in effect issuing directions to the Gateway to authenticate the Target on behalf of the client. Furthermore, the gateway is able to observe the request and response interaction with the Target, as the Gateway is in effect acting as a proxy for the client. Presumably, if these request and response transactions at the content level identify the identity of the end client, then the Gateway would be able to also reconstruct the combination of the identity of the end client and the identity of the URL being accessed, and furthermore the content of the subsequent transaction.

There is the potential for the Gateway and the Target to be operated by the same entity, but if the Target's identity can be derived from the Gateway's identity, then the entire purpose of the double layer encryption is lost, as the Relay host is then able to derive the combination of the client identity and the Target URL being accessed.

At this point I am a little perplexed by OHTTP. What is the point of this mechanism? The OHTTP mechanism can hide the combination of the identity of the client and the identity of the Target from the intermediate network elements, but so can any other form of VPN-like tunnel. The relative advantage of the VPN-like tunnel is that no other party is privy to the transaction itself, while in the relayed OHTTP model the Gateway Relay is acting as a proxy for the client and is privy to the subsequent transaction.

This arrangement can work adequately in simple transactional models of HTTP services but has some potential drawbacks in terms of trust as we've outlined. If the client wanted a greater level of assurance about the authenticity and privacy of its interactions with the Target and it also wanted a richer functional model of interaction that extended into reliable full duplex transport services, then once the initial connection is made between the client and the target, the communicating parties (client and target) need to take further measures. One possible measure is to initiate an end-to-end TLS session across these

relayed connections, treating the Relay and the Gateway as simple packet relays once the session has been initiated.

The basic framework of OHTTP (and ODoH for that matter) can be phrased as a bifurcated proxy model, where one part of the bifurcated proxy is only aware of the identity of the client, and the other part of the proxy is only aware of the identity of the target.

So can we take the more traditional view of a proxy (Figure 3) and combine it with this bifurcated proxy approach?

Figure 3 – A Proxy Intermediary

If we just look at simple proxy tools, then there is a rich history of such approaches, including SOCKS, where the client binds a local socket to a secure tunnel, tunnel-mode IPSEC, where IPSEC is used to operate a secure tunnel between two endpoints, TCP relay proxies, and the HTTP Connect Proxy. What if we wanted to leverage QUIC and TLS 1.3. with HTTP (HTTP/3) for such a proxy service? QUIC provides internal streams for multiplexing within the connection, and also supports datagram services. HTTP/3 builds on top of this a request/response behaviour, caching and authentication. This is the basis for the IETF's MASQUE effort.

MASQUE can extend the HTTP Connect service to use QUIC, tunneling stream-based and congestion-controlled transports like QUIC. QUIC can support in-network bandwidth aggregation of multiple access links, IP address translation to improve user privacy towards the server, and Link-specific congestion control. MASQUE can also be used to proxy any form Internet traffic over HTTP/3 by tunnelling IP through an HTTP server acting as an IP packet proxy gateway (Figure 4).

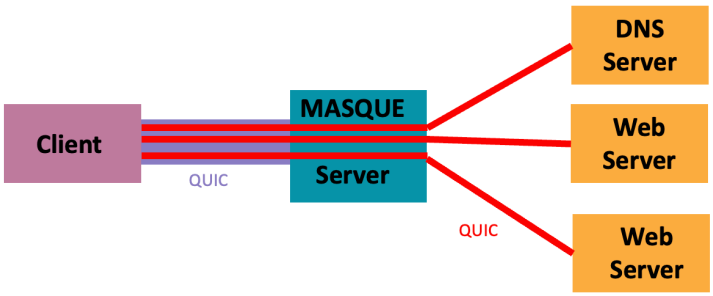

Figure 4 – MASQUE Proxy Intermediary model

The IP address used by the client in this setup is an address that is assigned by the MASQUE IP proxy rather than the "native" address of the client. If the client accesses a target URL then only the client and the IP proxy are aware of the original IP address of the client and the URL being accessed, which is not dissimilar to the outcome of the OHTTP approach.

The MASQUE IP Proxy is aware of the original address of the client and is aware of the IP address of the target servers that are being access by the client. If the TLS handshake did not use encrypted Client Hello, then the MASQUE IP proxy may even be aware of the URL being accessed. Aside from the client itself, no other party is able to assemble this same collection of information. The network between the client and the proxy will simply see an encrypted QUIC session between the end endpoints. The network

between the proxy and the target will see an encrypted session between the proxy and the target. The target will see a session between itself and the proxy-assigned IP address. However, the TLS session is an end-to-end session, and aside from the client and the target no other party is privy to the content that is passed between the two points.

This MASQUE approach strikes me as being a robust way to withhold gratuitous metadata from the network. The targets do not require any specific capability other than the support of TLS. The Masque proxy is not placed in the invidious position to being privy to the transactions between the client and the various targets. Yes, the MASQUE proxy does hold the endpoint information for each session, but not necessarily the actual URLs for the targets. In many ways this is just a change of wrapper for the SOCKS model, but QUIC does offer some further functionality to place multiple streams in the same client-to-Proxy QUIC session.

Can we apply the same OHTTP-styled bifurcated proxy to the MASQUE design?

This is a hot topic at present, and a number of vendors are positioning products here. One of the first such products into the mass consumer market is is Apple's Private Relay service.

The architecture of the Private Relay service is an OHTTP-styled approach as shown in Figure 5. The connection between the device and Relay 1 is based on QUIC and HTTP/3. Client HTTP transactions are passed through this dual relay framework and are passed along an end-to-end TLS session. All DNS queries are resolved in the same framework, using Oblivious DNS over HTTPS (ODoH).

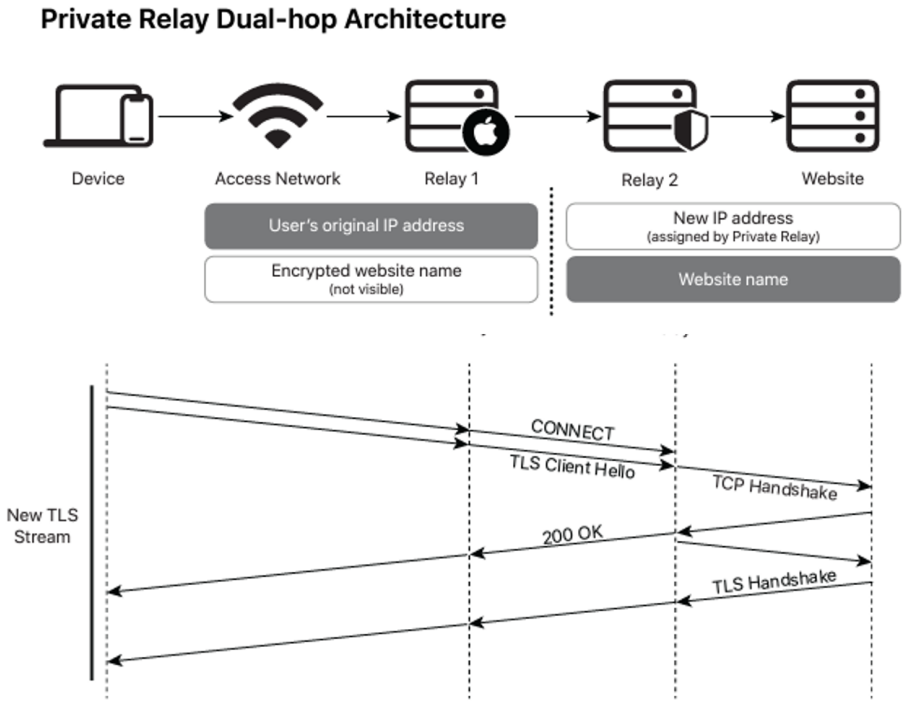

Figure 5 – Apple's Private Relay Architecture (from Apple's Private Relay Overview Documentation)

The entire concept of encapsulation and VPN tunnels is probably as old as the Internet itself and the issue is not intrinsically one of basic innovation in how to perform tunnelling. The effort at present appears to be directed towards integration of such approaches into public products, leveraging the capabilities of cloud providers operate such services across distributed platforms and offer services that can mask users' individual activities from network-based onlookers, yet still provide a robust and efficient user experience. The motivation here appears to be a desire to hide most aspects of user activity from the network carriage operators, and also from most forms of network intermediaries.

This approach certainly appears to be consistent with the directions for greater privacy espoused in RFC7258, but perhaps it may be useful to pause and think about the costs as well as the benefits of such measures for the Internet. The results of these actions, that create "privacy enclaves" in the Internet, are not unlike those of the enclosure of the commons in medieval England where communal-use fields were enclosed and were then only to be used by the designated land holder. The original open model of the Internet was one where each connected client could exchange packets with any other client without the need to involve intermediaries or seek permissions from a gatekeeper. In the name of enhanced privacy, we are now attempting to hide clients and their actions from the carriage network, from network intermediaries and common infrastructure elements, from each other and even from the platforms that host these client applications and from all other applications. Apple's approach with the Private Relay service has its counterparts in various product offerings from Google, Microsoft, Amazon, Cloudflare, to name a few of the more prominent players in this space. Oddly enough this activity, that is ostensibly intended to enhance users' privacy online, has not resulted in less volumes of personal data being gathered, nor has it allowed users to exercise greater levels of control as to how their data is trafficked and exploited. It is likely that this enthusiasm for greater levels of privacy in the Internet has been taken up by the major content and service providers as a means of protecting the data that their applications generate from being gratuitously collected by their competitors!

I may be overly cynical here, but I'm not sure as to what these obscuring methods really mean in terms of the larger picture of privacy protections for you and me as individual users of these services.

Disclaimer

The above views do not necessarily represent the views or positions of the Asia Pacific Network Information Centre.

Author

Geoff Huston AM, B.Sc., M.Sc., is the Chief Scientist at APNIC, the Regional Internet Registry serving the Asia Pacific region.

www.potaroo.net