Anatomy of a Web Request: The Process, its Pitfalls and the Trail it Leaves.[1]

Tim Watson

It's something we all take for granted. Browsing the Web seems so simple that it's almost as though the information we're accessing is sitting on our own computer, waiting for us to access it. So, how does clicking on a link or typing in a Uniform Resource Locator (URL) result in the page we request being displayed? And where in the chain of events are there opportunities for others to manipulate the process? Whether you're a home user concerned about privacy and security, a malicious attacker determined to undermine both, or a forensic investigator working hard to separate fact from fiction, a good understanding of this chain of events, the weak links and the trail of evidence created when surfing the Web is your best weapon. In this article, we will follow the journey of a single Web request from your browser to a webserver and back again. On the way we'll encounter pirates and hijackers, sinister African businessmen, deadly Sirens calling us to our doom, insidious poisons and Greeks bearing gifts. Hang on to your hats, it's going to be a bumpy ride.

We start our journey in the safe harbour of your personal computer. Against the odds, you've managed to keep your machine free from the myriad digital parasites that feed off your data and which could interfere with your actions. This is an important point. If the machine were to be infected with malware it would provide the first opportunity for an attacker to manipulate or fabricate your actions. But more of this later. For now, let's assume that your machine is clean[2].

Let's begin by opening up a browser and typing in a URL – for the sake of argument we'll use http://www.thedarkvisitor.com/. Whether you're using Firefox, Lynx, uzbl, Safari, IE or any other browser, when you enter this URL the program knows that it needs to send a Hypertext Transfer Protocol (HTTP) GET request to the associated webserver and that it will expect an HTTP response in return. Like any other application program, it uses the underlying operating system to do as much of the work as possible. Of the many system calls offered by the operating system's Application Programming Interface (API) – either offered directly by operating systems such as Linux or Mac OS X, or via functions in Dynamic Link Libraries (DLLs) if you're using Microsoft Windows – there are a collection of calls that interact with the operating system's networking subsystem. The browser uses these to prepare and to transmit its GET request, and then to wait for and to receive the response.

One reason why a browser feels so simple to use is because the user doesn't have to explicitly connect to the webservers being accessed. A URL just feels like a document name and accessing it seems as easy as accessing a document on your computer's

---

1 *This article originally appeared as the cover story for the first edition of Digital Forensics Magazine and is reproduced with the kind permission of the publishers.*

2 *We'll be making quite a few assumptions in this article but rather than list all the subtleties of each protocol and operating system data structure the approach will be to describe what typically happens when a user requests a Web page.*

hard disk. The complicated networking is hidden from the user by the browser, which itself is part of a massively distributed system.

If you think about it, there is no real difference between a file browser that searches for and fetches files on your computer and a Web browser, which does the same but is not limited to files on your computer alone. For a file browser to work it just needs to run its various program functions on one computer. A Web browser needs to ask programs running on other computers around the world to retrieve files on its behalf. At any one time there are millions of Web browsers and webservers talking to each other, joined by the common language of HTTP. In effect, there is one large software application that spans the globe and part of it is running on your computer.

In an application program that just runs on one computer, information is passed back and forth between the program's functions by putting it in areas of shared memory that all the functions can read from and write to. In a distributed system, the same information flows are needed but the computers involved don't share the same memory and so they need another way of exchanging information, which they do by encapsulating the application information in packets that are sent from one part of the system to another over a network. So our browser needs to package up its GET request into one or more packets and to send them to the webserver by asking the operating system to do so. But where exactly should it send the data and how will the response find its way back? We need a destination and a return address.

Addressing – Location, Location, Location

In fact, there are six addresses involved. To see why, it's best to consider a real-world situation that has many parallels with accessing things over the Internet. Let's consider delivering a package to someone living in a block of flats.

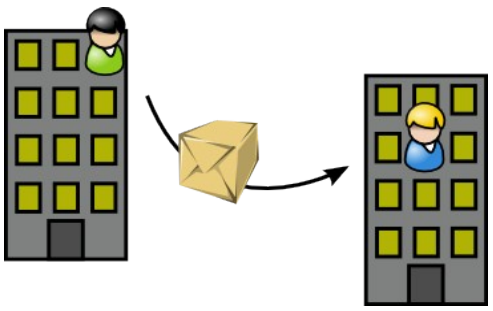

Figure 1. Delivering a package

We can see from Figure 1 that delivering a package is easy. Straight from the sender in one building to the recipient in another. But nothing is ever this easy. Which building? Which street, city, country? When the delivery arrives at the intended building, which of the buzzers should be pushed to get the recipient on the intercom so that they can accept the package? And if something needs to be sent in return, where should it go?

We use a postal address to get the package to the right building and a flat number to identify which buzzer to press. Although we only need to supply the destination address, the mail service will transfer our package from pickup van to train, perhaps an aeroplane, one or more sorting offices and eventually into a delivery van. Each step of this journey requires that the package is put in a container and sent on to the next stage. Consequently the package is repeatedly boxed, addressed to the next stage and transferred to a differently addressed container as it continues on its way. And when it arrives at the correct building, if the wrong buzzer is pushed, or if the intended recipient isn't listening for the buzzer, then the package will not reach its destination.

So we need a postal address and flat number together with a correctly addressed container for the first hop of the journey (which will be repeatedly replaced). Three addresses: one for the building, one for the person in the building and one for the next stage of the journey. If we require a reply then we need another three addresses to ensure that the response can find its way back again. Consequently, in our analogy as in a Web request, we require six addresses.

Let's see how these six addresses relate to our Web request. The journey that the request takes from your machine to the webserver takes it through several computers. On my computer, to reach http://www.thedarkvisitor.com/, there are eighteen computers in the journey. For the destination address, we have to specify which computer on the Internet to send it to and which of the programs listening for network input on that computer should be given the request. These are the IP address and the port number, respectively. Armed with the destination IP address, each computer along the way can determine the next hop on from itself and can forward the packet by putting it inside another package, called a frame, which uses another address – a Media Access Control (MAC) address – to uniquely identify the network interface of the next hop computer. So, we need three addresses: port number, IP address and the MAC address of the first step of the journey, and we'll need another three addresses to ensure that the response gets back to us. Your operating system will take care of the return addressing and it will also supply the MAC address of the first step of the journey for us as soon as we tell it the destination IP address but we don't have that yet. To get it, your browser will need to use another distributed system. The Domain Name System (DNS).

DNS – The Internet Telephone Directory

Your browser has to construct another request – a DNS request – to ask a DNS server to resolve the name of the webserver into an IP address. To understand DNS it's best to think of it as working like an online telephone directory. You look up the name of the person you want to contact and the directory supplies you with the number that you need to use to contact them over the (telephone) network. Over the Internet, it's IP addresses rather than telephone numbers that you need so that you can contact others.

The IP address of at least one DNS server has been configured on your computer, either statically or dynamically when the computer was powered up by making a

Dynamic Host Configuration Protocol (DHCP) request for a DHCP server to give it an IP address, details of one or more DNS servers to use, which gateway to use to access external networks etc. Since most personal computers are configured to use DHCP this is another common point at which an attacker can strike, supplying details of a rogue DNS server or gateway that will then resolve the webserver names you send into the IP addresses of malicious webservers, unrelated to the intended destination.

Let's recap. We want to visit http://www.thedarkvisitor.com/ and your browser needs a destination IP address. To get this it has to send a request to a DNS server so that the name of the webserver we want to access is resolved to its IP address. The IP address of one or more DNS servers is stored on your machine and the browser retrieves the first one. DNS servers listen on User Datagram Protocol (UDP) port 53 so the browser constructs a UDP segment addressed to port 53. Hang on, I hear some of you cry, a segment? We've had packets and frames and now a segment, what do they all do? Well, a segment is simply another type of digital package: segments contain application data and are addressed to ports (i.e. they are addressed to a process on the destination computer with a return address of a process on the originating computer). As Figure 2 shows, segments are placed inside packets, which use IP addresses to identify which machines are communicating, and a packet is repeatedly placed inside and then removed from a succession of frames as it travels from machine to machine, making its way from its source to its final destination.

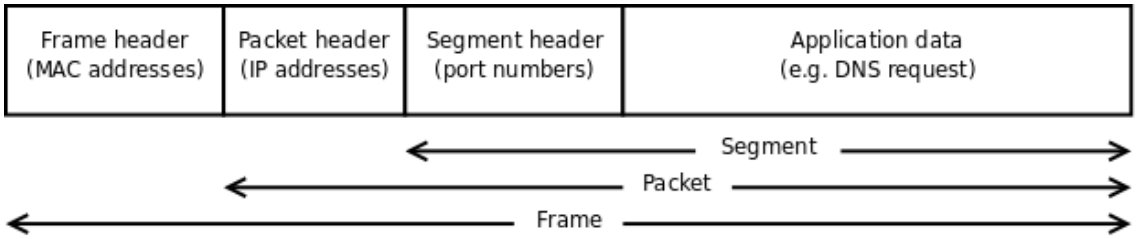

Figure 2. Data encapsulation

As we follow the journey from browser to webserver and back again, we're concentrating on three things: how does the entire process work, what opportunities are there for an attacker to subvert the process and where might we find forensically significant artefacts. We've mentioned that an attacker might use malware to manipulate the Web request process on the originating host – every forensics investigator worth their salt knows about the Trojan defence and the importance of showing that submitted evidence takes into account any malware on the system – and it hardly bears mentioning that there will be a number of pieces of information on the host to help reconstruct what has gone on there. But now, in pursuit of an IP address for the destination webserver, we are about to send a DNS request inside a segment, in a packet and enveloped in a frame over the network (see Figure 3). Just as every forensic investigator knows where to look for evidence on a seized computer, we also need to understand the forensic artefacts created by network activity that exist outside of the originating computer. If an attacker can convince your machine to wrap its packet in a frame addressed to his computer, he can intercept the packet, look inside and

modify it if necessary before forwarding it on to its destination. He can also spoof a reply. If we don't know where to look for evidence, how will we be able to counter a defence based on the malicious interception of network traffic? From the safety of your computer, we're about to enter the dangerous, coastal waters of your LAN and the wild seas of the Internet.

Frame 1 (82 bytes on wire, 82 bytes captured)

Ethernet II, **Src MAC: 00:11:d8:0c:0a:36, Dst MAC: 00:22:3f:5c:60:66**

Internet Protocol, **Src IP: 192.168.1.2, Dst IP: 192.168.1.1**

User Datagram Protocol, **Src Port: 52598, Dst Port: 53**

Domain Name System (query)

Transaction ID: 0x2963

Flags: 0x0100 (Standard query)

Questions: 1

Answer RRs: 0

Authority RRs: 0

Additional RRs: 0

Queries

www.thedarkvisitor.com: type A, class IN

Figure 3. Example DNS query captured using wireshark packet analyser

Your browser has constructed a packet ready to be sent to the DNS server. Your operating system inspects the destination IP address and tries to match it against an entry in its routing table – a data structure held in RAM that tells the operating system which network interface to use and which machine to send it to for the next stage of the journey – and if it can't find a match an error will be generated. Consequently, the routing table is forensically significant as it determines what a machine will do with a packet, assuming the host firewall doesn't interfere in any way (and thus the firewall configuration is also important when reconstructing events). On a typical PC, a route is added to the routing table when a network interface is configured or when a gateway is defined to allow access to external networks, however, most of the intermediate machines in a journey across the Internet will be routers and they have dynamic routing tables that change as routers talk to each other using a variety of routing protocols. Again, subverting these protocols or the protocols used by switches to construct loop-

free paths through networks is another way for an attacker to redirect packet flows and leap on board the convoy of data, cutlass in hand. Network logs, if they exist, can reveal both the protocol conversations and any attempts to subvert them, and switch and router configurations can add useful corroborating evidence. Most network devices have the ability to log network traffic and events and it is good practice to monitor and log network traffic as it travels across a network. Two standard, open-source tools are commonly used for this – tcpdump (http://www.tcpdump.org/) and wireshark (http://www.wireshark.org/) – and becoming competent in their use is an essential skill for all network forensic investigators.

So, if everything is configured correctly, our DNS request's destination IP address will match a row in the routing table and the operating system will know which network interface to use and which machine to send it to. The operating system needs to wrap the packet in a frame and address it to the correct machine. To do so it needs the relevant MAC address and we need to get to grips with yet another protocol. What started as a simple Web request is rapidly turning into cascade of interrelated actions, each one with weaknesses that can be exploited and a variety of associated pieces of evidence. Let's continue on our journey.

ARP – Asking for Directions by Shouting and Listening

Within your operating system is another RAM-based data structure called the ARP cache. The Address Resolution Protocol (ARP) is used to enable computers to find out which MAC address should be associated with a given IP address. Recent results from ARP requests are stored in the ARP cache.

From an attacker's perspective, ARP is a wonderful protocol: a computer broadcasts a request asking for the MAC address associated with a particular IP address and whoever responds to the requesting computer is believed by it. You can even send a computer a new MAC address for a given IP address at any time and it will just believe you. An analogy would be a banking protocol for paying your credit card bill. You walk into a bank, grab a paying-in slip and ask in a loud voice what account number to use to send money to your credit card company. Whatever gets shouted back – you don't even check to see if it came from the shifty looking character with a below average leg count, eye patch, idiosyncratic hand replacement and pet parrot – you believe the reply and promptly fill out the slip.

Hacker-friendly programs such as ettercap exploit this protocol vulnerability to poison the ARP caches of a victim machine and its gateway so that all traffic in both directions is redirected through an attacker's machine (see Figure 4). This Man in the Middle (MITM) attack can be used to make it look as though the victim has been doing things that they shouldn't or to gain unauthorised access to systems or data. The deadly Siren calls that lure unwary traffic towards malicious servers can be identified as suspicious ARP requests and responses within network logs, as in Figure 5, or in the contents of ARP caches (before ettercap cleans up after itself) and can be very useful when determining whether the evidence is consistent with this type of network tampering.

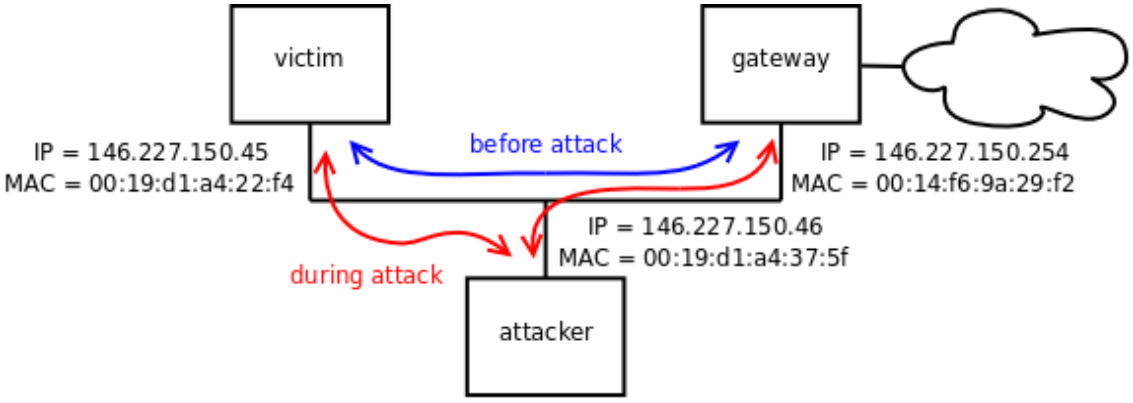

Figure 4. ARP poisoning in action (network diagram)

As mentioned previously, Figures 4 and 5 show ettercap ARP poisoning in action. Figure 4 shows the network used and Figure 5 shows the packets captured using tcpdump. Working through the packets shown in Figure 5, the first four packets show the attacker (IP address ending with 46) requesting the MAC addresses of the victim and gateway and the associated responses. The next eight packets are malicious ARP replies not associated with any specific request that target the victim and gateway and tell each of them that the other's MAC address is 00:19:d1:a4:37:5f, which just happens to be the attacker's MAC address. These packets are retransmitted every ten seconds to ensure that the ARP caches stay poisoned. The final two packets show ettercap cleaning up after itself by using the same unrequested reply trick to correct the ARP caches on the victim and gateway.

17:08:42.918718 arp **who-has** 146.227.150.**45** **tell** 146.227.150.**46**

17:08:42.919203 arp **reply** 146.227.150.**45** **is-at** 00:19:d1:a4:22:**f4**

17:08:42.928903 arp **who-has** 146.227.150.**254** **tell** 146.227.150.**46**

17:08:42.930510 arp **reply** 146.227.150.**254** **is-at** 00:14:f6:9a:29:**f2**

17:08:43.940085 arp **reply** 146.227.150.**254** **is-at** 00:19:d1:a4:37:**5f**

17:08:43.940093 arp **reply** 146.227.150.**45** **is-at** 00:19:d1:a4:37:**5f**

...

17:08:47.983375 arp **reply** 146.227.150.**254** **is-at** 00:19:d1:a4:37:**5f**

17:08:47.983390 arp **reply** 146.227.150.**45** **is-at** 00:19:d1:a4:37:**5f**

17:08:57.993502 arp **reply** 146.227.150.**254** **is-at** 00:19:d1:a4:37:**5f**

17:08:57.993525 arp **reply** 146.227.150.**45** **is-at** 00:19:d1:a4:37:**5f**

...

17:09:58.055349 arp **reply** 146.227.150.**254 is-at** 00:19:d1:a4:37:**5f**

17:09:58.055372 arp **reply** 146.227.150.**45 is-at** 00:19:d1:a4:37:**5f**

17:09:58.107659 arp **reply** 146.227.150.**254 is-at** 00:14:f6:9a:29:**f2**

17:09:58.107672 arp **reply** 146.227.150.**45 is-at** 00:19:d1:a4:22:**f4**

Figure 5. ARP poisoning in action (packet capture)

Ettercap

One of the standard weapons in the hacker's armoury is ettercap, a powerful program that performs Man in the Middle (MITM) attacks and extracts passwords from network traffic. Ettercap uses ARP poisoning to redirect switched traffic to an attacker and can retrieve passwords from many protocols, including HTTP, IMAP 4, POP, FTP, RLOGIN, SSH1, TELNET, VNC, ICQ, IRC, MSN, NFS, SMB, MySQL, X11, BGP, SOCKS 5, LDAP, SNMP, HALF LIFE and QUAKE 3.

Ettercap can inject, remove and modify packet contents based on selection criteria supplied by the attacker and it can be used to sniff SSL secured data by presenting a fake certificate to victims. An attacker can use ettercap to passively monitor network traffic and to collect detailed information about hosts on the network (operating system used, open ports, running services, etc.). In active mode, it is easy to kill any network connection by selecting it from a list, perform DHCP and DNS spoofing, view a victim's Web browsing activity in real time and much more. As a network security monitoring tool, ettercap can be used to spot ARP poisoning attempts and whether anyone is sniffing every packet on the network.

Ettercap is open source and freely available for all major operating systems including Linux, Windows and Mac OS X. See the official website at https://www.ettercap-project.org/

DNS Revisited – Trust No One

After determining the correct MAC address to use, your operating system can now send out the DNS request and receive a reply. If your computer is using an older or badly configured wireless LAN then dangers lurk within weak encryption mechanisms and a rogue wireless access point could be used by attackers to lure you to your doom, another deadly Siren call that can be hard to resist. Even if the network is trustworthy, the response from the DNS server may not be, as the protocol and software used by DNS servers to exchange information can be exploited to poison the DNS cache. This is a particularly deadly attack as it is so difficult for the users to detect. If you carefully type in the URL of your favourite online bookshop and your ISP's DNS server responds to the browser's DNS request with an IP address, how do you know whether the IP address is correct? Poisoning a DNS server's cache is known as pharming and is a far more effective method of collecting credit card details, login credentials and personal information than it's more prevalent but less potent relative, the phishing attack, as seen in the many fake bank security emails and also in the equally popular 419 scams from the relatives of deposed African leaders. Even if the DNS server is untouched, it is still possible to spoof a DNS response, especially if the transaction ID is not very random, something that can happen if the operating system is virtualised, as virtualisation can reduce the randomness of random number generators. This is also a problem for Initial Sequence Numbers (ISNs) used to keep network conversations safe from session hijackers, which we discuss shortly.

So, at last, as soon as we receive our DNS response we're ready to send out the HTTP GET request. We have the destination IP address and since we are using HTTP your browser will use destination port 80 unless we tell it to use a different port by including it after a colon in the URL (e.g. http://www.example.com:8080/). Your operating system will supply the MAC address of your gateway. Our return details should also be included: a source IP address, a source port number (chosen at random by the operating system) and a source MAC address, (which will be included in the frame header). However, unlike the previous DNS request that used UDP, an HTTP request uses the Transmission Control Protocol (TCP) and before we see our GET request arriving at the destination webserver and a response coming back we need to understand a little more about TCP.

TCP – Getting Connected

If I want to send you a short message about my Caribbean sailing holiday I'll use a postcard. This is UDP – a method of communicating by sending one or more individually addressed messages that can arrive in any order, possibly with some missing. If I want to send you a draft of my large travel book, sending each page on a separate postcard would be confusing. Pages will arrive out of order and some may well be missing. You would have to spend time receiving postcards, sorting them, requesting duplicates of the missing ones etc. TCP solves this problem, allowing you to setup a connection and then TCP does all the sorting and requesting redeliveries and even transforms the contents of individual packets into a stream of data. TCP is like a logical hosepipe – the sender shoves data in one end and the receiver sees it in the

same order coming out of the other end. Since HTTP is often used to transfer large webpages that won't fit into a single packet it is natural for it to use TCP.

Unlike UDP, which simply sends a packet whenever someone has any data to transfer, TCP has three phases. First it sets up the connection using a 'three-way handshake', then it uses the connection to allow both parties to exchange data (i.e. the same connection is used both to send and receive data by each endpoint) and finally the connection in both directions is closed down.

The three-way handshake is shown in Figure 6, which is from the Request For Comment (RFC) 793 that describes TCP. The essence of TCP is that each endpoint computer uses a number to show how much data it has transferred so far. If I tell you that I've sent 30 bytes and you acknowledge that you received 30 then everything is fine. If I then say that now the total transferred is 40 bytes but you say that you still only have 30 bytes from me then TCP knows to retransmit the missing data. If the numbers get too far out then TCP will reset the connection and another will have to be set up. The numbers are called sequence numbers and an initial sequence number (ISN) is chosen at random. This protects against a delayed packet arriving later and interfering with another connection, and protects against an attacker guessing the number and forging a TCP packet, which would allow them to hijack the TCP connection (known as a session and hence this attack is a TCP session hijack, not to be confused with cookie stealing session hijacks).

Following the packets in Figure 6 we can see that machine TCP A sends a packet with a control bit set to show that it is a synchronisation packet – a SYN packet – that synchronises the connection by sending an ISN. Line 2 shows a reply packet from TCP B that acknowledges A's ISN and sends one of its own, hence this packet is a SYN-ACK packet. Finally, the connection is setup on line 3 when A sends an ACK to acknowledge B's ISN.

	TCP A				**TCP B**
1	**SYN-SENT**	→	<SEQ=100> <CTL=SYN>	→	SYN-RECEIVED
			<SEQ=300> <ACK=101>	←	**SYN-RECEIVED 2**
	ESTABLISHED	←	<CTL=SYN,ACK>		
3	**ESTABLISHED**	→	<SEQ=101> <ACK=301>		
			<CTL=ACK>	→	ESTABLISHED

Figure 6. TCP three-way handshake (adapted from RFC 793)

Consequently, every HTTP GET request we make is sent by setting up a TCP connection between your computer and the destination webserver – we send a SYN,

the webserver sends us a SYN-ACK and then we complete the connection with an ACK. The next packet sent on this connection contains our GET request and subsequent packets from the webserver will also be part of this TCP connection and will contain the webpage we requested or a response containing an error message.

If our simple Web request to http://thedarkvisitor.com/ is similar to the test request that I have just run from my computer, it will have resulted in 7 TCP connections to four different webserver IP addresses (the main webpage contains images that have to be retrieved from other webservers) with 235 packets transferred that together contained 80,216 bytes. Assuming that, like my computer, yours is 17 hops away from the destination webserver, each of the 235 packets will have been placed in 17 different frames as they travelled between source and destination and many ARP request/reply frames will have been generated as these machines updated their ARP caches. A reasonable estimate is that this one Web request resulted in well over 4,000 frames being transmitted.

If we hope to keep our Web browsing private we can think again – 17 other computers know what we requested and what was returned, since the contents of our connections are unencrypted, and every computer on each of the 17 networks we passed through could potentially use a MITM attack to view the contents too.

In this article, we have followed the journey of a Web request from its origins as a URL typed into a browser, through the operating system of the source computer and seen it pass through several intermediate machines on its way to the destination webserver. We've seen how ARP and DNS are involved in the route that it takes and how a request is sent over a TCP connection that first has to be synchronised by a three-way handshake. What appears to most users as a simple click of the mouse turns out to be a very complicated process involving thousands of frames and many networks.

With so many opportunities for malicious intervention and manipulation, evidence of a particular Web request or response is not necessarily the smoking gun it might appear to be. But any request creates ripples in the digital ocean that travel widely and can be spotted, if you know what you're looking for and where to look.