THE BASIC OPERATION OF A WEBPAGE

Stephan Dalal[*]

CITE AS: 1 GEO. L. TECH. REV. 235 (2017)
https://perma.cc/U6XE-YD6S

INTRODUCTION

The world is changing fast. At the end of 2015, nearly 44% of the world population had access to the Internet.[1] By some reliable estimates, the remaining 56% of the world's population will be connected to the vast digital network by 2025.[2] In less than nine years, every human will have the potential to be connected to another, no matter the physical separation. This makes online privacy a particularly critical endeavor, placing privacy professionals at the center of a critical mass of initiatives, services, responsibilities, and leadership. For example, the scale of the U.S. federal government's digital and IT mission is staggering, as the U.S. federal government represents a North Star to so many private and public institutions, the leadership and knowledge of privacy professionals working in our government can impact the privacy and technology landscape in ways that are orders of magnitude larger than they would be in other institutions.[3] Nonetheless, the impact of many private entities that operate in the collection and processing of data can have an enormous impact on how notions of privacy are shaped.[4]

Interaction with our network today largely takes places through the use of a web browser, which is a type of software that allows a user to access and

[*] Technology Editor, GLTR; Georgetown Law, J.D. expected 2017; Columbia University, M.S. 2013; Ramapo College of New Jersey, B.A. 2010. © 2017, Stephan Dalal.
[1] *Internet Users (Per 100 People)*, THE WORLD BANK, http://data.worldbank.org/indicator/IT.NET.USER.P2?end=2015&locations=1W&start=2015&view=bar (last visited Mar. 29, 2017) [https://perma.cc/Y98V-EB5Y].
[2] *See* European Dialogues, *European Dialogue: Eben Moglen & Jule Goikoetxea*, YOUTUBE (June 20, 2016), https://youtu.be/hnVSSQctF0s [https://perma.cc/BD9N-UBPN].
[3] See for example http://pusle.cio.gov, which maintains a directory of the web presence of all federal government assets.
[4] For example, Facebook, Google, or Apple, all of which have had pivotal roles in redefining the notions of online privacy.

interact with an ever-growing number of hosted services.[5] Some of these services,[6] like Google, NYTimes.com, or FederalRegister.gov may be well known to a large proportion of internet users, while other less prominent services, are less well known. Hosted services play an important role in shaping how we receive, interact with, and share data through the web. But how exactly does a web browser work, and in which ways are privacy implicated, if at all, in the context of our interaction with the web?

This brief guide aims to provide the reader with a foundational understanding of the main components of the internet through practical examples and demonstration. The explainer will begin by explaining the function of a computer server, and how a web browser interacts with and intermediates the flow of data to and from a computer server. Next, we descend into a brief study of one of the most fundamental parts of a computer server, the access log. We take a request, break down its components, and digest the practical significance of each major portion. Finally, we engage in a technical discussion of the implications of third-party hosts and tracking methodologies that are in wide use, and rely on the foundational structure in the way computers speak with one another. This explainer is not intended to answer all of the questions that exist in this space. Rather, it aims to help answer the most foundational ones, and empower the privacy professional with the requisite knowledge to begin *asking* the difficult questions that will be needed to help shape our future.

THE BASIC OPERATION OF A WEBPAGE

Understanding the basics of how a webpage loads is essential to understanding how the connected world works. This is especially true when the task at hand might include the need to seek out and identify tracking on a website or service, explain a certain digital mechanism or objective to a policymaker or technical expert, or to understand the structure of our digital world to begin to think about ways to build useful tools.

[5] "Hosted" means available to the outside world accessible through the Internet. *See* OXFORD ENGLISH DICTIONARY, https://en.oxforddictionaries.com/definition/host (last visited Mar. 29, 2017) [https://perma.cc/X636-ZDDC].

[6] In this piece I sometimes refer to "webpage" or "webservice," which are wholly different. Nonetheless, for our purposes the distinction between the two is not necessarily important. Both exist in a publicly available form on the internet, which users often access through a browser. So when I use these terms, I use it to mean either a web page or a web service that can be accessed through a web browser.

A Server Is Software

A webpage or web service is hosted on a server, which is a specially configured computer. The basic understanding of a "computer server" usually includes an image of *some* physical computer, with different colored flashing lights, perhaps sitting in the concrete basement of some remote facility, and distinct from the computers we interact with daily. But the term "server" refers to the function of a specially configured computer rather than as a description of its physical features. This function is a simple one: to *serve* data to some other computer that has requested it. At a fundamental level, computer servers are no different than the one you have used to access GLTR's website, and the one you used to access this document.

The part of a computer server that often makes it distinct from a personal computer is its software.[7] This may seem like a distinction without a difference, but it is a fundamental one. Physical components of a computer are directed to engage in tasks through a set of instructions, such that we can alter *what* tasks the computer engages in and *how* those tasks are carried out by altering the declarations to the computer. Because server software[8] directs a computer to act in a certain way, we could presumably use this type of software on any type of computer that had the ability to interpret and execute the instructions. If you wanted to, you could turn your personal computer into a computer server with the right software. Throughout this piece we will examine the log file of the world's most widely used webserver software, Apache.[9]

Server software has three core features: the ability to handle requests by other computers; to store *and* organize files in a certain way; and to make those files accessible to the outside world, here through the internet. Because server software is task-intensive on certain physical components of a computer, the computer server does often require certain hardware components to perform its function. This is the reason why our internalization of a computer server may not be wholly inaccurate. Imagine the hard drive on your computer, and now imagine trying to store the entire photo albums of 1 billion users: you would need many hard drives, and you would likely need a large facility to store your specialized computer. Aside from the practical reality, the software controlling a system that large would not be categorically

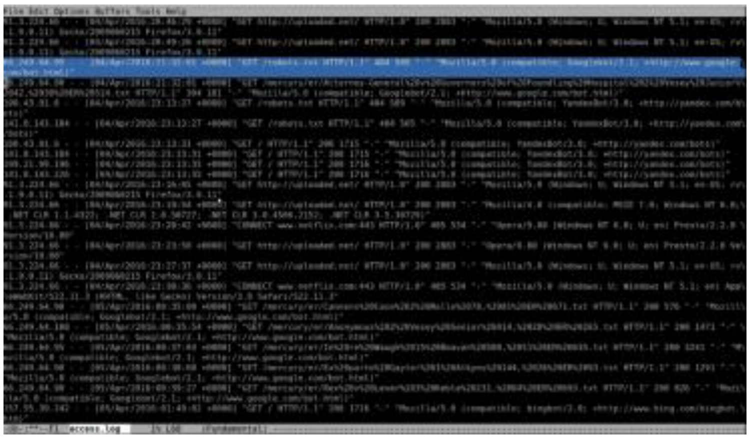
[7] *See About Apache*, APACHE SOFTWARE FOUND., https://www.apache.org/foundation/ (last visited Mar. 29, 2017) [https://perma.cc/MK9M-TH8V].
[8] I use the term "software" to mean a set of a series of instructions that a computer is able to understand and execute.
[9] *See About Apache*, *supra* note 7.

different from server software that you could install on your personal computer.

To better explain how computer servers communicate with and process requests, consider the other meanings of the word "server," such as a server at a restaurant—which we also may refer to as a waiter or waitress. We can think about a computer server's function in terms of a waiter at a restaurant. A server greets you when you sit down at the table, takes your order, brings you your food, and concludes the interaction by bringing you a check. We think of waiters who work at restaurants not in terms of their physical differences but in terms of their function as a waiter—i.e., a waiter is a waiter not because of their physical construction—although like computer servers, some physical attributes may aid waiters in performing some of their tasks—but because being a waiter is dictated by the actions they take to fulfill that function. Recognizing the distinction that a server is software (and not hardware) is critical. Like a waiter, a computer server performs analogous tasks but in a digital context.

We now must make one small shift in our language. Up until this point we have discussed computer servers. We discussed the fundamental distinction between a computer server and your personal computer, and introduced the analogy of a waiter at a restaurant to explain that the function of a computer server is dictated by the instructions given to it by the server software. Because we agreed that software is the defining feature of a computer server, and that in theory computer in computer server could be replaced by your computer or some bulky computer that spans the length of a large facility in a remote location, we can replace computer in computer server with web. We will now be referring to a webserver, and its functions. Here, web in webserver means that our computer server has now been connected to a network of other computer servers, and that any user can access some of the content on this webserver through a browser. Nothing fundamental has changed. We're simply moving from one type of computer server to another type, and this type happens to be a computer server that can handle requests from the outside world and display the product of those requests through a computer browser. Now that we've established this new terminology, let's discuss what, exactly, happens when you visit a webpage.

Before you even visit a webpage (the restaurant), your browser's request has to be routed to the server. In our physical-world analogy, this would be like driving, biking, or walking to the restaurant along *some* route that led to the restaurant. Routing your browser's request to the address of a webserver is a complex process that is outside the scope of this Explainer. You only need to recognize that this is itself a separate process. Instead of the physical-world street address a restaurant may use to identify its location to patrons, directory services, such as domain name service ("DNS"), look for servers based on their internet protocol ("IP") address.



```
[illegible]
```

Once you arrive at the webserver (or restaurant), the server greets your browser by asking it a few preliminary questions and stores the responses to those questions in a log file.[10] Unlike a physical-word visit to a restaurant, the questions your browser is asked to answer are obligatory. While the exchange can be thought of as a polite question-and-answer between your browser and webserver, the server has to ask these questions because it knows nothing about who is making the request. The server may need to know the type and version of your browser, the types of operating system your browser was built for, and other pertinent information. Humans are remarkable at inferring information from context. Computers do not possess these same features, and so take nothing for granted, and must ask questions in order to form some basic understanding about who is making the request and how it ought to respond.

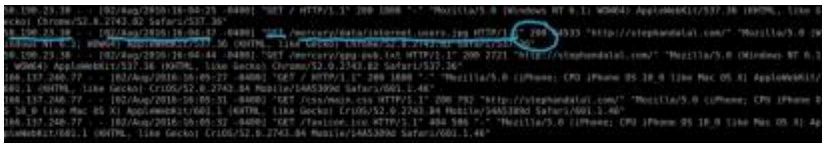
Below is an example of the record one type of server software, Apache, makes during your arrival. In Apache this preliminary information is stored in a file called *access.log*. This access.log file was generated by my personal webserver and contains a listing of every request that was made to my webserver. Notice the highlighted line in blue. This line displays an entry showing that a Googlebot visited my website on April 4, 2016. It also stores other information about this particular visit, such as the type of browser/software that was used to make the request (here Googlebot's own

[10] The questions and storage may vary based on the type of server software the webserver is using.

special software version 2.1), the location of the content requested, and the time and date of the request.

Accordingly, here is how we would read the Googlebot's visit. On April 4, 2016, at 9:32 p.m. (UTC), a service identifying itself as Googlebot/2.1, made a GET-request for a text file on [this server] called *robots.txt*. The request originated from the IP Address: 66.249.64.95, and the file *could not be found*.[11] At the time this request was made, I did not have a robots.txt file[12] on my webserver. So naturally, my server could not deliver that document to the Googlebot.

We can analogize that interaction back to the restaurant. When your waiter asks you what you'd like to eat, you tell them the name of the dish you'd like to order. In effect, you are making a request to the waiter. You may tell your waiter that you'd like to order the salmon with asparagus, the waiter will acknowledge your request, put your order in to the chef, and will fulfill your request by bringing you your meal. When you visit a webpage, your browser is responsible for making a request to the webserver for content. For example, if you are visiting the New York Times webpage, your computer may ask for a series of articles that make up the front page. This digital menu presented to your browser—and then to you—like the menu at a restaurant, enables you to select the content you would like to access and communicate that request to the server. You browse the articles that seem interesting, and when you've found one that looks appetizing, you click on a link to the article that asks the New York Times webserver to load the files that contain the full article.[13] And this same process takes place on this website. Here is another example of a request for an image on my webserver.



```
[illegible] - - [02/Aug/2016:16:04:25 -0400] "GET / HTTP/1.1" 200 [illegible] "-" "Mozilla/5.0 (Windows NT 6.1; WOW64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/52.0.2743.82 Safari/537.36"
[illegible] - - [02/Aug/2016:16:04:37 -0400] "GET /mercury/data/internet_users.jpg HTTP/1.1" 200 [illegible] "http://stephandalal.com/" "Mozilla/5.0 (Windows NT 6.1; WOW64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/52.0.2743.82 Safari/537.36"
[illegible] - - [02/Aug/2016:16:04:44 -0400] "GET /mercury/[illegible].txt HTTP/1.1" 200 2721 "http://stephandalal.com/" "Mozilla/5.0 (Windows NT 6.1; WOW64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/52.0.2743.82 Safari/537.36"
[illegible] - - [02/Aug/2016:16:05:27 -0400] "GET / HTTP/1.1" 200 [illegible] "-" "Mozilla/5.0 (iPhone; CPU iPhone OS 10_0 like Mac OS X) AppleWebKit/601.1 (KHTML, like Gecko) CriOS/52.0.2743.84 Mobile/14A5309d Safari/601.1.46"
[illegible] - - [02/Aug/2016:16:05:31 -0400] "GET /css/main.css HTTP/1.1" 200 [illegible] "http://stephandalal.com/" "Mozilla/5.0 (iPhone; CPU iPhone OS 10_0 like Mac OS X) AppleWebKit/601.1 (KHTML, like Gecko) CriOS/52.0.2743.84 Mobile/14A5309d Safari/601.1.46"
[illegible] - - [02/Aug/2016:16:05:32 -0400] "GET /favicon.ico HTTP/1.1" 404 [illegible] "-" "Mozilla/5.0 (iPhone; CPU iPhone OS 10_0 like Mac OS X) AppleWebKit/601.1 (KHTML, like Gecko) CriOS/52.0.2743.84 Mobile/14A5309d Safari/601.1.46"
```

[11] I can tell that the file was not found (and thus not delivered to the Googlebot) because of a transaction code that appears immediately after the file requested. Here, we see "GET /robots.txt HTTP/1.1" 404. If you've ever visited a webpage and received the error message *Error 404 – Requested Page Not Found*, that transaction was likely stored on some log file as a 404-error for your request.

[12] Google's crawler that visited my webpage is following appropriate etiquette by making this request. Many webservers store a simple text file called *robots.txt* that contain a list of rules for where, when, and how often a crawler may visit the given webserver. For example, in the *robots.txt* file, one can "DENY" a crawler the permission to traverse the website.

[13] A story published on the New York Times website may have images associated with the story. These images are themselves files, and so the webserver has to load many files in order to present you with one coherent *thing*.

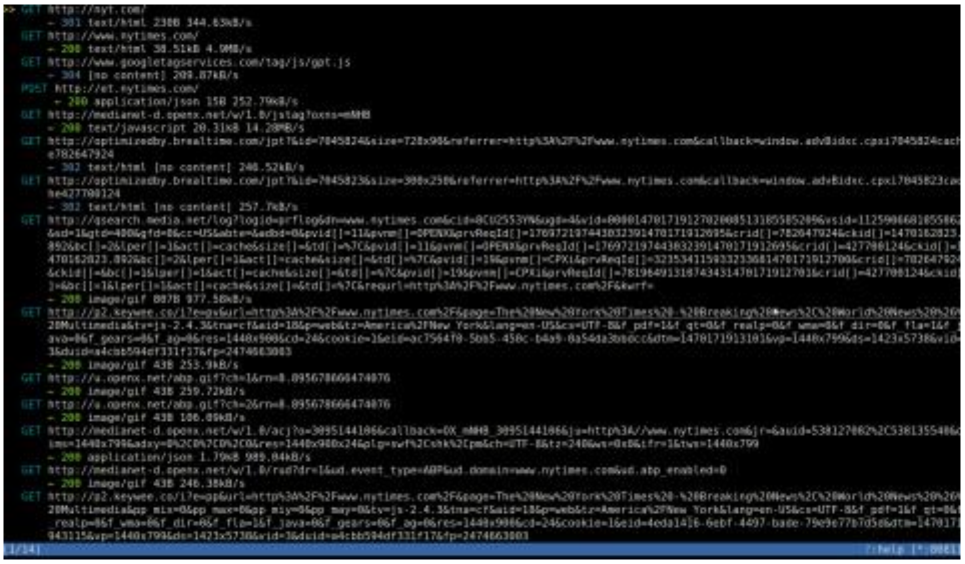
A natural language phrasing of this request could be: *Hi, StephanDalal.com, I'm 50.190.23.30. GET me the document called "internet.users.jpg," please.* Using our physical-world analogy, I was greeted, took a look at the menu, placed an order (made a request) with the waiter (server), and my food brought to me (my request fulfilled). Notice the circled transaction code, 200. This indicates that the request was, unlike the request for robots.txt, successfully fulfilled.

Notice the type of information the server collected about my request: my location, which is based on my IP address; when I accessed the information; what browser software I used and its version; and what I was accessing.[14] Unlike visiting a restaurant, when you visit a webpage, your browser will make several requests to the webserver each second. Recall our earlier discussion about the natural state of computers where nothing is taken for granted. Everything on a webpage needs to be "served" to you, from text, to pictures, to videos, other services, tracking scripts, as well as many other features that a web service may use.

You can start to imagine that if given a list of 10, or 50, or 100 requests, one could begin to piece together *what* a person was looking for, and perhaps even *who* they were. Would it be unreasonable to infer that a person who made three requests: a request to a bar exam preparation website; a request to the bar application page for the State of New York; and a request to a website that provided information about how to obtain a notary, was a law student, perhaps graduating in the near future? What if the IP address showed those requests originating from Washington, DC? Would you be comfortable inferring that this person was a student at Georgetown, George Washington, or American University? What if I gave you a fourth request: to web page that sells tickets for Georgetown basketball games? How confident would you be in inferring that this person is very likely a law student at Georgetown?

Below is a more detailed example of the conversation between my web browser and the webserver hosting one of the New York Times' webpage. You should see a few familiar terms in this list of requests, like the 'GET' marker. Each GET entry here represents a piece of content the browser has asked the New York Times webserver to serve to it. You may also notice the 'POST' marker, which is the name of a protocol that performs a function opposite to a GET request—you can think of the POST request as <u>post</u>ing a sign to a tree: you are expressing a message to someone (or to the world).

[14] Here I am accessing an image which shows a chart of the total number of internet users, but in other contexts a user may be accessing a political essay, information about a personal medical condition, or information that most people may consider sensitive.



```
>> GET http://nyt.com/
       ← 301 text/html 238B 344.63kB/s
   GET http://www.nytimes.com/
       ← 200 text/html 38.51kB 4.9MB/s
   GET http://www.googletagservices.com/tag/js/gpt.js
       ← 304 [no content] 209.87kB/s
   POST http://et.nytimes.com/
       ← 200 application/json 15B 252.79kB/s
   GET http://medianet-d.openx.net/w/1.0/jstag?oxns=mNHB
       ← 200 text/javascript 20.31kB 14.28MB/s
   GET http://optimizedby.brealtime.com/jpt?&id=7045824&size=728x90&referrer=http%3A%2F%2Fwww.nytimes.com&callback=window.advBidxc.cpxi7045824cach
       e782647924
       ← 302 text/html [no content] 240.52kB/s
   GET http://optimizedby.brealtime.com/jpt?&id=7045823&size=300x250&referrer=http%3A%2F%2Fwww.nytimes.com&callback=window.advBidxc.cpxi7045823cac
       he427700124
       ← 302 text/html [no content] 257.7kB/s
   GET http://qsearch.media.net/log?logid=prflog&dn=www.nytimes.com&cid=8CU2553YN&ugd=4&vid=0000147017191270200051318558520996vsid=11259066810558
       &sd=1&gtd=400&qfd=0&cc=US&abte=&adbd=0&pvid[]=11&pvnm[]=OPENX&prvReqId[]=1769721974430323914701719126958crid[]=782047924&ckid[]=1470162823.
       892&bc[]=2&lper[]=1&act[]=cache&size[]=&td[]=%7C&pvid[]=11&pvnm[]=OPENX&prvReqId[]=176972197443032391470171912695&crid[]=427700124&ckid[]=1
       470162823.892&bc[]=2&lper[]=1&act[]=cache&size[]=&td[]=%7C&pvid[]=19&pvnm[]=CPXi&prvReqId[]=3235341159332336814701719127008crid[]=78264792
       &ckid[]=&bc[]=1&lper[]=1&act[]=cache&size[]=&td[]=%7C&pvid[]=19&pvnm[]=CPXi&prvReqId[]=781964913187434314701719127018crid[]=427700124&ckid
       ]=&bc[]=1&lper[]=1&act[]=cache&size[]=&td[]=%7C&requrl=http%3A%2F%2Fwww.nytimes.com%2F&&wrf=
       ← 200 image/gif 887B 977.58kB/s
   GET http://p2.keywee.co/1?e=pv&url=http%3A%2F%2Fwww.nytimes.com%2F&page=The%20New%20York%20Times%20-%20Breaking%20News%2C%20World%20News%20%26
       20Multimedia&tv=js-2.4.3&tna=cf&aid=18&p=web&tz=America%2FNew York&lang=en-US&cs=UTF-8&f_pdf=1&f_qt=0&f_realp=0&f_wma=0&f_dir=0&f_fla=1&f_j
       ava=0&f_gears=0&f_ag=0&res=1440x900&cd=24&cookie=1&eid=ac756470-5bb5-450c-b4a9-0a54da3bbdcc&dtm=1470171913101&vp=1440x799&ds=1423x5738&vid=
       3&duid=a4cbb594df331f17&fp=2474663083
       ← 200 image/gif 43B 253.9kB/s
   GET http://u.openx.net/abp.gif?ch=1&rn=0.895678666474076
       ← 200 image/gif 43B 259.72kB/s
   GET http://u.openx.net/abp.gif?ch=2&rn=0.895678666474076
       ← 200 image/gif 43B 106.09kB/s
   GET http://medianet-d.openx.net/w/1.0/acj?o=3095144106&callback=OX_mNHB_3095144106&ju=http%3A//www.nytimes.com&jr=&auid=538127082%2C5381355486
       ims=1440x799&adxy=0%2C0%7C0%2C0&res=1440x900x24&plg=swf%2Csk%2Cpm&ch=UTF-8&tz=240&ws=0x0&ifr=1&tws=1440x799
       ← 200 application/json 1.79kB 989.04kB/s
   GET http://medianet-d.openx.net/w/1.0/rud?dr=1&ud.event_type=ABP&ud.domain=www.nytimes.com&ud.abp_enabled=0
       ← 200 image/gif 43B 246.38kB/s
   GET http://p2.keywee.co/1?e=pp&url=http%3A%2F%2Fwww.nytimes.com%2F&page=The%20New%20York%20Times%20-%20Breaking%20News%2C%20World%20News%20%26
       20Multimedia&pp_mix=0&pp_max=0&pp_miy=0&pp_may=0&tv=js-2.4.3&tna=cf&aid=18&p=web&tz=America%2FNew York&lang=en-US&cs=UTF-8&f_pdf=1&f_qt=0&f
       _realp=0&f_wma=0&f_dir=0&f_fla=1&f_java=0&f_gears=0&f_ag=0&res=1440x900&cd=24&cookie=1&eid=4eda1416-6ebf-4497-bade-79e9e77b7d5e&dtm=147017
       943115&vp=1440x799&ds=1423x5738&vid=3&duid=a4cbb594df331f17&fp=2474663083
[1/14]                                                                                                                              ?:help [*:8081]
```

But much more data lies beneath these request logs. The above communication is *not* an Apache access.log file. Instead, it was generated using a program called MITM-Proxy.[15] Earlier, we looked at an access log that had unique entries about visitors to a webpage, much like a guest book at a restaurant; MITM-Proxy records the visit from a different perspective. The proxy used to generate this image stands between the web browser and the webserver, like the manager of a restaurant who may be able to monitor the communication that is occurring between the waiter, patron, and chef, as a silent observer. We could imagine the manager of the restaurant, to ensure that wait staff communicated politely and efficiently with customers, began to make a sentence-by-sentence log of each communication between parties. Like that, a MITM-Proxy displays (and records) each-and-every public[16] communication that takes place between the webserver and web browser. The result is a precise listing of the requests made, the *protocol* or *method* used to handle the request, and who (or to whom) the requests (or responses) were directed. Thus, the listing above represents all of the public communication

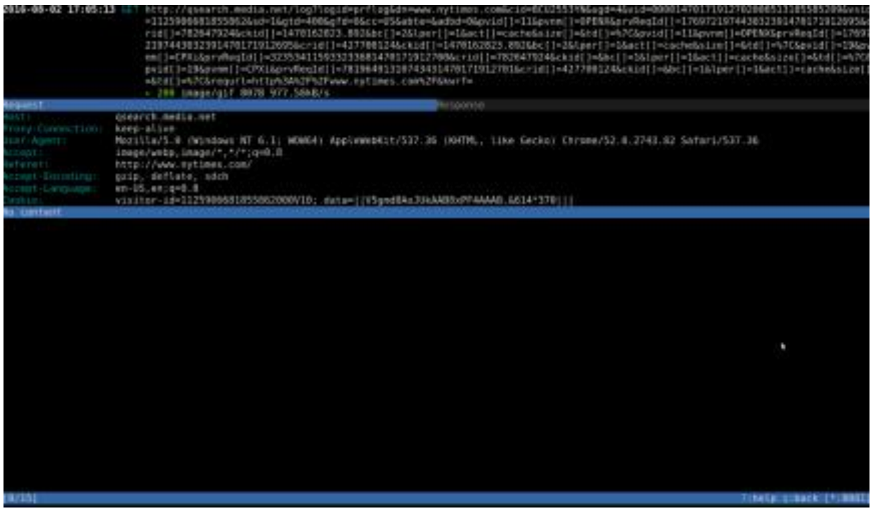
---

[15] MITM in MITM-Proxy, stands for Man in the Middle. *How Mitmproxy Works,* MITMPROXY, https://mitmproxy.org/doc/howmitmproxy.html (last visited Apr. 18, 2017) [https://perma.cc/K5RP-WZNA].

[16] Under normal operation, MITM-Proxy would be unable to listen-in on communication sent using HTTPS, which is a secured internet communication protocol that encrypts web traffic. *See* Sang Ah Kim, *HTTPS: Staying Protected On the Internet*, 1 GEO. L. TECH. REV. 199 (2016), https://www.georgetownlawtechreview.org/wp-content/uploads/2017/01/Kim-1-Geo.-Tech.-L.-Rev.-119-2016.pdf [https://perma.cc/F4US-M82B].

that occurred between my web browser and the New York Times' webserver to deliver me one piece of content.

What is important to understand is that these requests occur between your browser and a webserver numerous times per second. A request to my webserver for simple content like an image of a graph may only require one or two request/response transactions. Often however, the request/response transaction between a web browser and webserver—which is merely a conversation between two computers—includes hundreds (and potentially thousands) of requests that are fulfilled within seconds, as the complexity of the web service or content your browser is requesting increases. If a webserver is properly configured, connected to the outside world, and accepting traffic, each individual request will *usually* be met with an individual response.



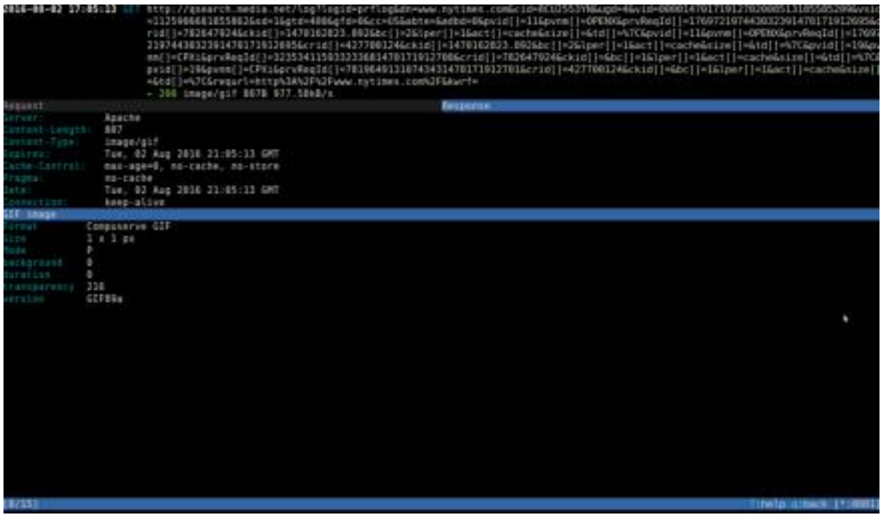
```
2016-08-02 17:05:13 GET http://qsearch.media.net/log?logid=prf&ogdn=www.nytimes.com&cid=[illegible]
                        [illegible]
                        [illegible]
                        [illegible]
                        [illegible]
                        [illegible]
                        =&td[]=%7C&requrl=http%3A%2F%2Fwww.nytimes.com%2F&wurf=
                        ← 200 image/gif 8078 977.58kB/s
Request                                                    Response
Host:                qsearch.media.net
Proxy-Connection:    keep-alive
User-Agent:          Mozilla/5.0 (Windows NT 6.1; WOW64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/52.0.2743.82 Safari/537.36
Accept:              image/webp,image/*,*/*;q=0.8
Referer:             http://www.nytimes.com/
Accept-Encoding:     gzip, deflate, sdch
Accept-Language:     en-US,en;q=0.8
Cookie:              visitor-id=[illegible]; data=[illegible]
No content
```

Looking at one of the request/response transactions from our example, we can see a detailed response to the request my browser made.

This request seems to be from a third-party service, called *qsearch*, that the New York Times relies on. The New York Times may be using *qsearch* to help keep track of its content, or to perform some web site analytics.[17] Why was a request made to search if we did not make any request through our browser?

[17] I take no position on any normative questions this example may raise, such as whether a web service *should* or *should not* use third-party services. Many third-party services provide a legitimate and valuable service for content providers. Instead, this Technology Explainer focuses on unearthing the existence of parties that may be involved in any given web communication. I leave it to the experts, policymakers, and professionals working in this

Importantly, your browser will, on its own, load all of the coded content on a webpage when it first visits the page. For example, when you decided to visit GLTR's website, you simply typed in the domain name. When your browser was routed to GLTR's server, it had to answer a series of questions, and once that was completed, GLTR presented your browser with all of the files needed for your browser to display our website to you. Those files contained instructions for your browser, in a language called Hypertext Markup Language (HTML).[18] When you first visit a webpage by instructing your browser to make a request to the web server, the web server automatically assumes that your browser will want to load the initial (home page) content to you. At times, this initial phase can include tracking scripts, analytic code, or instructions from the webserver to load other files that live at different web servers.



```
2016-08-02 17:05:13 GET http://qsearch.media.net/log?logid=prflog&dr=www.nytimes.com&cid=[illegible]
                        [illegible]
                        [illegible]
                        [illegible]
                        [illegible]
                        [illegible]
                        =&td[]=%7C&requrl=http%3A%2F%2Fwww.nytimes.com%2F&kwrf=
                        ← 200 image/gif 807B 977.58kB/s
Request                                                   Response
Server:          Apache
Content-Length:  807
Content-Type:    image/gif
Expires:         Tue, 02 Aug 2016 21:05:13 GMT
Cache-Control:   max-age=0, no-cache, no-store
Pragma:          no-cache
Date:            Tue, 02 Aug 2016 21:05:13 GMT
Connection:      keep-alive
GIF image
Format         Compuserve GIF
Size           1 x 1 px
Mode           P
background     0
duration       0
transparency   216
version        GIF89a

[8/15]                                                    ?:help q:back [*:8081]
```

Now, take another look at the detailed response pictured above. Can you discern just from reading the report what *type* of content we received? Notice the 'Content-Type' field, and you will see that we received an image,

space to make meaningful and important considerations about the use and disclosure of such services.

[18] This is the primary language of the web browser, and contains instructions for your browser, which range from simple instructions like *Bold This Text*, to more complex instructions like load: script.js.. The .js file extension denotes that the file is a JavaScript file, which is a sophisticated tool used by certain webpages or web services to deliver content or intermediate information between a user's browser and the webserver hosting the webpage or service.

specifically a GIF.[19] Perhaps the report contains other interesting information about this response? We can look to the following fields for more information: *Format, Size, Mode, Background, Duration, Transparency, Version*. Looking to the *Size* field, we notice that the image is quite small—in fact, the image is only 1 x 1 pixels in size. An image of this small size surely has no practical value, as far as pictures go. So what was the point of serving us this image? Here, we've found what's called a tracking-pixel, a mechanism that can be used to track visitors to a web service. How these pixels work are saved for another day.[20] But recall our discussion of the access log file, which we said tracks several pieces of information about a request that is made to a web server. Also recall the entry on my server's access.log file about my request for an image of a graph. Once I made that request, my webserver recorded my visit. Would the size of a requested image change that? The answer is no. And so one of the ways a tracking-pixel works, is by causing a practically invisible image to be *requested*, logged by the web server that stores that nearly invisible image, and then *respond* with the invisible image.

Like tracking physical footsteps , web servers are able to follow your activity on the web. After all, they have stored in their log a request you made, the time you made, and where you were when you made it. One or two request logs may not lead to a meaningful model of where you went on the web or what you were doing. But what if we loaded 30 or 40 pieces of content, each of which made a request to some web server that recorded it, and we continued to load these files at each web site you visited. We could paint a detailed portrait of the visitor, and perhaps learn information about their lives. You were able to do that with four pieces of information I gave you earlier about the law student.

Understanding how a web server processes requests and, understanding what kind of information log files detail about g a user's browser history, are key to understanding fundamental legal and policy discussions surrounding privacy in the twenty-first century. In the criminal

[19] A Graphics Interchange Format (GIF) is a type of image format that is widely used across the internet.

[20] Although outside the scope of this guide, an invisible pixel works by causing your browser to load the image file (even though there is no image to view). When your browser loads the link associated with the pixel, your browser must make a request to the party hosting the pixel, and as we've discussed, when your browser makes a request to a webserver it must answer some questions and provide the webserver with information about your session. This information can later be used to correlate your visit to the originating site—i.e. the webpage where the pixel script existed—and the third party site hosting the pixel. The more web services that contain these pixels, the more data about your web browsing behavior can be generated.

domain, judges struggle with applying the Fourth Amendment's third party doctrine, which states that a user does not have a reasonable expectation of privacy in information surrendered to a third party, to digital interactions that are almost invariably required to go through a third party service provider in order to occur..[21] While this explainer takes no position in those debates, a technologically correct understanding of the issues at hand is crucial to their successful resolution.

CONCLUSION

We learned some useful information about the basic structure of how a web page loads, and the foundational components involved in the transaction: the web browser, web server, access.log file, and a series of request/response transactions. Web technologies are built on the foundations of these mechanisms. Understanding the basic structure of how services communicate allows those with the prerogative to influence sensible changes to our laws and policies the opportunity to shape the design of systems that recognize our data-sharing world. Because our technological landscape is in flux, the process of answering key legal and policy questions will become increasingly reliant on an understanding of foundational technologies, like the web.

[21] *See* Smith v. Maryland, 442 U.S. 735 (1979) (holding that an individual does not have a reasonable expectation of privacy in the things they convey to another person); United States v. Jones, 565 U.S. 400 (2012) (holding that the warrantless location monitoring of a suspect's movement over the course of a month violated the Fourth Amendment, but because the government's surveillance techniques involved an impermissible trespass); Riley v. California, 134 S.Ct. 2473, 2479 (2014) (observing that in the context of warrantless searches of cell-phones seized incident to a lawful arrest, "Our cases have historically recognized that the warrant requirement is an important working part of our machinery of government," and that while "our decision today will have an impact on the ability of law enforcement to combat crime . . . [p]rivacy comes at a cost.") (internal quotations omitted).