Client Monitoring with HTTPS

Felix Hartmond, Simon Bauer*
*Chair of Network Architectures and Services, Department of Informatics
Technical University of Munich, Germany
Email: felix.hartmond@tum.de, bauersi@net.in.tum.de

***Abstract*—With the increasing number of users and devices on the internet, it is more and more important for an administrator to know who and what is using his networks and services. Therefore it is necessary to find out information about the clients. This process is called fingerprinting. With the ongoing deployment of HTTPS information from the application layer can no longer be read directly. In this paper, we will look at the different layers of the HTTPS protocol stack and examine which fingerprinting approaches are still possible, or even only possible, with the added encryption.**

***Index Terms*—https, client fingerprinting**

1. Introduction

1.1. Motivation

With the growing number of users and devices on the internet is it more and more interesting who and what is producing traffic. Information about the users and devices is helpful for a lot of things. For example, it can be used to optimize services and networks. It can also be used to improve the security of a system as it can help to identify potential malicious actors. Such actors can also be tracked with such information which allows finding out more about them.

Getting information from network traffic is getting harder with the extended use of the encrypted version of the HTTP protocol: HTTPS. But there are far more ways to find out information about a client than just reading plain HTTP requests. And, even if encryption makes the analysis harder, it does not make it impossible. In this paper, we look at the different layers of the network stack and examine different ways to analyze a client with information provided by the different layers.

1.2. Outline

In Section 2, we will give a rough overview of different kinds of fingerprinting. After this, we will look at the involved protocols in an HTTPS communication and analyze which kinds of fingerprinting are possible on the different layers. We will go down the network stack from top to bottom and look at each layer. For each layer, we will analyze which methods were developed to gather information about the client from the information provided by this layer.

We will start at the very top and look at the Hyper Text Transfer Protocol (HTTP) in Section 3. Then, we look at the Transport Layer Security protocol (TLS), which is the new layer in HTTPS, compared to HTTP, in Section 4. After this, we examine in Section 5 the Transmission Control Protocol (TCP), which is usually used by HTTP connections as transport protocol [7] together with the Internet Protocol (IP). We look at these two layers at once as many approaches use information from both protocols. Finally, we will take a brief look at transport layer protocols. As these protocols are not used end to end they have some limitations we will also look at.

In the end, in Section 7, we will recap which information can be gathered overall.

2. Fingerprinting

Fingerprinting can be split into two categories: device fingerprinting and user fingerprinting. The methods used for fingerprinting can also be categorized into active methods and passive methods.

Device fingerprinting is about gathering information about the device which is communicating. These things include the operating system, drivers, protocol implementations or browsers. These can be very interesting to understand which device types are active in the network and which software in which versions they use. This information can be relevant to be able to optimize services and network infrastructure for the way how they are used. User fingerprinting is about getting information about the user who is actively using the applications. This can be interesting for tracking a user across different devices or services to be able to optimize, for example, the presented content for this user.

Fingerprinting methods can be split up into two categories: Active and passive methods. Active methods actively send probes to a device and analyze the responses. These methods require the ability to send probes to a device but are very mighty if there is no firewall in front of the device. They are mighty because due to the active participation of the observer it is possible to send specifically crafted packets to the client. On the other side, passive methods do not send out any new traffic, they just analyze traffic which is passing. Such methods completely rely on the traffic which is already there. But they has the huge advantage that they do not have to happen real time. It is possible to just capture and store traffic and do the complete analysis afterwards or even running an analysis on a traffic capture which was not stored with fingerprinting in mind.

3. HTTP

If we would not use HTTPS, the information from the application layer would tell us a lot about the client and its action. There would be a lot of HTTP headers sent with every request, for example, the User Agent, which explicitly tells which system is used by the client. Additionally, we would be able to see all the requests including all parameters which gives us very detailed information about user and his actions. But, HTTPS hides all this application data through the addition of encryption. For the encryption an TLS layer is added between the TCP layer and the HTTP layer. This layer acts as an container for the HTTP layer and encrypts all data from it. So, theoretically, all data from the HTTP layer should not provide any information to an observer. But a lot of research was done to find out if it is possible to gain information about a client despite the presence of encryption.

Stöber et. al. [19] took a look at mobile applications and tried to find out which application observed traffic belongs to. They were able to identify the application with a success probability of 90% despite not seeing the content of the traffic. For they approach they utilized that a lot of application traffic is not directly triggered by a user action. Instead, most traffic is done by the application is the background on a reglar basic. They analyed the patterns of this background traffic and were able to deduce the running application from the traffic patterns.

Zion et. al. [13] tried to find out the client's operating system, the used browser and the application from encrypted traffic. They used supervised machine learning to assign labels of the form (OS, Browser, Application) to the packets. They were able to successfully classify packets with their approach. This shows that is is possible to get basic information about a user by only looking at the encrypted traffic.

Panchekno et. al. [14] analyzed if it is possible to find out which website was accessed if the traffic is protected and anonymized by anonymization networks like TOR. For their approach, they used a support vector machine (another machine learning technique) and were able to archive a true positive rate of 73% for a false positive rate of 0.05%.

Also, Cai et. al. [2] looked at encrypted traffic in the context of defenses like Tor. They used a simple model of network behavior to find out which homepage is accessed by a user and were able to identify the requested page with a good success rate.

Overall research has shown that encryption can not completely hide information from the communication. A very impressive example of getting information out of encrypted packets was presented by Wright et. all. They took a look at Voice over IP communication and were able to reconstruct spoken text from encrypted traffic. This was possible as the audio was encoded with a codec with variable frame rate which caused network packets of variable length. These varying packet lengths were sufficient to reconstruct spoken phrases. [22]

Dyer et. al. [4] took a conceptional look into the problem of hiding information completely through encryption or obfuscation. They came to the conclusion that it will always be possible to extract some kind of information from encrypted traffic as long as bandwidth optimizations are done, what every protocol does.

4. TLS

The Transport Layer Security Protocol takes care of encrypting all data above the TLS layer. To be able to do encryption the protocol has first to do a handshake between client and server. As there are no shared secrets between client and server initially, the first messages of the handshake are unencrypted. In the beginning, the client sends a so called "hello message" in which it tells the server about its version, cipher suites, compression methods and extension they support. The only other unencrypted message from the server contains the servers public key. After this, no more messages from the client are unencrypted. [3]

As often multiple homepages are hosted on a single server and these homepages use different certificates, TLS needs to know the requested homepage already during the handshake to provide the correct certificate. As ordinarily the domain is presented first after the finished handshake TLS has an extension called Server Name Indication (SNI). This extension presents the requested domain in the client hello message of the TLS handshake to make it possible for the server to present the correct certificate for the requested domain afterward. But the information from this is very limited as only the domain can be retrieved but neither any further information about the request nor details about the requesting client. [6]

To get more information about the client from the client hello message, Martin Husák et. al. went through the different values included in the client hello message and analyzed them towards the differences in the values for different client types. They found out that the most values are quite similar for different client types, but they found one very interesting value: the list of supported cipher suites. This list of support cipher suites differs enough between different clients to make it possible to distinguish them. [10]

Martin Husák et. al. used two methods to build a codebook which maps cipher suite lists to user agents. For the first one, a server based one, they logged the cipher suite lists as well as the user agent of incoming https connections on a web server. This method produces very accurate results but the clients have to visit a specific server to be included in the codebook. Because of this, they combined the fist with a second method which analyzed network traffic which includes HTTP as well as HTTPS connections from the same clients. By matching the connections from the same client together they were able to extend the codebook with clients which did not connect to their servers. In their tests, the top 10 cipher suite lists covered 68.5% of the network traffic and the top 31 cipher suite lists covered 90% of the traffic. Over their measurements, they discovered 1598 different cipher suite lists. [10]

Despite a very good variance in the cipher suite list there were still multiple user agents which correspond to a single cipher suite list. So they used a tool which extracts information about a system from a user agent like browser name, operating system or vendor. With this, they were able to classify the traffic into several categories for devices (desktop, mobile, unknown) as well as applications (browser, command line, application, update, unknown). [10]

5. TCP/IP

The Transport Control Protocol (TCP) takes care of the reliability network connection. It takes care of retransmission of lost packets, flow control, congestion control and multiplexing of connections between two hosts. [17]. The Internet Protocol (IP) provides addressing between the hosts which communicate with each other. Also, packet routing is done by this network layer. [16]. As many fingerprinting approaches use information from these two layers together, we will look at these two layers together.

The TCP specification, as well as the IP specification, does only describe the scenarios of regular operation. They do not specify corner cases like how an implementation should behave in the case of packets which should never occur, like strange combinations of flags or if TCP segments or IP fragments overlap. As these scenarios are not specified, different implementations have different behavior when processing such packets. This can be used for active fingerprinting. By sending such impossible packets the used implementation of the communicating host can be examined. For example, the tool NMAP implements a test where a packet with a FIN flag is sent without establishing a connection beforehand. Even if this packet should be just discarded, some implementations send an answer to it. [18]

In addition to implementation differences caused by edge cases and implementation bugs, TCP implementations also have slight differences in their behavior concerning retransmissions and congestion control. Different implementations have slightly different retransmission timeouts. By sending a TCP SYN packet, and measuring the time between the SYN-ACK packet and his retransmissions it is also possible to detect the used implementation as Veysset et. al. showed. [21]

Many more people have done research on active TCP/IP fingerprinting and developed tools which can execute the proposed measurements and classifications. Grek Taleck took a very detailed look at the different aspects of a TCP/IP implementations and developed a tool called SYNSCAN which's "objective is to fingerprint every aspect of a TCP/IP implementation". [20] Ofir Arkin and Fyodor Yarochkin developed Xprobe2 which uses an approach based on confidences to find a result, instead of relying on an exact match with a known fingerprint. [1]

All methods mentioned for TCP/IP so far were active methods. But an active method needs either an accessible device or a connection from this device to a controlled server which then can send probes as replies. Having an accessible device is rather easy for servers but in general, clients are hidden behind a firewall which makes it impossible to send probes to them. Passive fingerprinting methods allow fingerprinting just by analyzing the traffic from a client at any point on the path between client and server.

Different researchers created tools which analyze initial values of different fields from the TCP and the IP header. The siphone tool was a proof-of-concept tool which only analyzed the TCP window size, the IP Time to Live and the IP Don't Fragment bit from packets of a TCP connection. The p0f tool and the ettercap tool look especially at the SYN packet of the TCP connection. Ettercap additionally analyzes the SYN-ACK packet. In addition to the features of siphone, they also analyze different options and flags from the TCP header. [12] [23]

Vern Paxson took a different approach and developed a passive fingerprinting tool called tcpanaly. This tool analyzes a complete flow and looks at the congestion control behavior of the implementation as different implementations differ in their rounding of TCP's sstresh. It also takes a look at response delays for the creation of acknowledgments and watches for occurrences of a bug caused by uninitialized variables. [15]

TCP has a timestamp option which is intended for round trip time measurements. Kohno et. al. showed that it is possible to use the values from this timestamp option to detect clock skews which can be used to uniquely identify a user. Even if these variations are very small, they found out that it is possible to use that technique even if the observer is quite far away from the client. [11]

6. Link Layer Protocols

When looking at network layers below the IP layer, fingerprinting on these layers has a different initial situation - it is link dependent as first the IP Protocol takes care of the end-to-end addressing of a packet. So, when looking below the IP Layer we no longer look at packets, instead look at frames which only exist for one hop in the connection. This means, to be able to monitor a client we have to be very close, especially on the same link as the client. So fingerprinting on these layers is not possible for server administrators who want to know something about the clients sending requests. But for administrators who want to learn about the clients in their local network, these methods are useful.

Fingerprinting on the Link Layer is of course highly dependent on the used Link Layer protocol. When looking at an Ethernet frame the header only contains the source and destination MAC addresses, a type id of the content of the next higher layer and an optional VLAN tag. One of the few things which can be derived from these header fields is the vendor of the sender's network card as the MAC address contains an "Organization Unique Identifier" (OUI). [5] But the network card vendor does not give any reliable information about the software running on the client. Despite that, MAC addresses can be modified by a user or in case of virtual machines are not bound to physical hardware at all.

But, this situation is different for 801.11 networks. 802.11 is far more complicated than ethernet as it has connection establishment and authorization mechanisms, so it has far more potential to reveal information about the client.

Franklin et. al. used the active scanning for driver fingerprinting. Every 802.11 client periodically sends probe requests to discover access points in range. The time intervals between the probes can be used to identify the used driver. A huge advantage of this approach is that no specialized equipment is required for the measurements as the probes can be received any ordinary 802.11 hardware. [8]

It's even possible to get a step further and look at the analog signal created by an 802.11 interface. Gerdes et. al. took this approach and found out that a device can be identified and tracked by small differences caused by hardware and manufacturing inconsistencies. With this approach, it is not possible to identify the used driver or operating system, but it allows to identify a user on the link layer even if he changed his MAC address to another value or moves to another network. [9]

7. Conclusion

We have seen that even if the presence of encryption makes the analysis of the application layer harder it does not block fingerprinting. Despite the encryption, it is still possible to find out a lot about the actions of the user on the application layer. Especially with the current development in machine learning approaches, it is very hard to hide information from the application layer completely. The added TLS layer provides encryption but offers with the supported cipher suite list a way itself to fingerprint the device.

On the IP and TCP layers, there is also fingerprinting possible due to implementation differences. As these layers are below the TLS layer, they are completely untouched by the added encryption. Even lower on the link layer, there are also ways to do fingerprinting but these are dependent on the used link layer technology on this link. Additionally, the observer has to be on the same link as the client to do observations on the link layer.

Overall the Protocol stack of HTTPS has a lot of possibilities which can be taken into account when specific information about the clients should be examined. As fingerprinting methods are possible at very different aspects of a connection a specific attacker model is needed to say if a setup is sufficiently protected against fingerprinting.

References

[1] O. Arkin and F. Yarochkin. Xprobe v2.0 - a "fuzzy" approach to remote active operating system fingerprinting. 08 2002.

[2] X. Cai, X. C. Zhang, B. Joshi, and R. Johnson. Touching from a distance: Website fingerprinting attacks and defenses. In *Proceedings of the 2012 ACM Conference on Computer and Communications Security*, CCS '12, pages 605–616, New York, NY, USA, 2012. ACM.

[3] T. Dierks and E. Rescorla. The transport layer security (tls) protocol version 1.2. RFC 5246, RFC Editor, August 2008. http://www.rfc-editor.org/rfc/rfc5246.txt.

[4] K. P. Dyer, S. E. Coull, T. Ristenpart, and T. Shrimpton. Peek-a-boo, i still see you: Why efficient traffic analysis countermeasures fail. In *2012 IEEE Symposium on Security and Privacy*, pages 332–346, May 2012.

[5] D. Eastlake. Iana considerations and ietf protocol usage for ieee 802 parameters. RFC 5342, RFC Editor, September 2008. http://www.rfc-editor.org/rfc/rfc5342.txt.

[6] D. Eastlake. Transport layer security (tls) extensions: Extension definitions. RFC 6066, RFC Editor, January 2011. http://www.rfc-editor.org/rfc/rfc6066.txt.

[7] R. T. Fielding, J. Gettys, J. C. Mogul, H. F. Nielsen, L. Masinter, P. J. Leach, and T. Berners-Lee. Hypertext transfer protocol – http/1.1. RFC 2616, RFC Editor, June 1999. http://www.rfc-editor.org/rfc/rfc2616.txt.

[8] J. Franklin, D. McCoy, P. Tabriz, V. Neagoe, J. Van Randwyk, and D. Sicker. Passive data link layer 802.11 wireless device driver fingerprinting. In *Proceedings of the 15th Conference on USENIX Security Symposium - Volume 15*, USENIX-SS'06, Berkeley, CA, USA, 2006. USENIX Association.

[9] R. M. Gerdes, T. E. Daniels, M. Mina, and S. Russell. Device identification via analog signal fingerprinting: A matched filter approach. In *NDSS*, 2006.

[10] M. Husák, M. Cermák, T. Jirsík, and P. Celeda. Network-based https client identification using ssl/tls fingerprinting. In *2015 10th International Conference on Availability, Reliability and Security*, pages 389–396, Aug 2015.

[11] T. Kohno, A. Broido, and K. C. Claffy. Remote physical device fingerprinting. *IEEE Transactions on Dependable and Secure Computing*, 2(2):93–108, April 2005.

[12] R. Lippmann, D. Fried, K. Piwowarski, and W. W. Streilein. Passive operating system identification from tcp / ip packet headers *. 2003.

[13] J. Muehlstein, Y. Zion, M. Bahumi, I. Kirshenboim, R. Dubin, A. Dvir, and O. Pele. Analyzing https encrypted traffic to identify user operating system, browser and application. 03 2016.

[14] A. Panchenko, L. Niessen, A. Zinnen, and T. Engel. Website fingerprinting in onion routing based anonymization networks. In *Proceedings of the 10th Annual ACM Workshop on Privacy in the Electronic Society*, WPES '11, pages 103–114, New York, NY, USA, 2011. ACM.

[15] V. Paxson. Automated packet trace analysis of tcp implementations. *ACM SIGCOMM*, 27, 07 2000.

[16] J. Postel. Internet protocol. STD 5, RFC Editor, September 1981. http://www.rfc-editor.org/rfc/rfc791.txt.

[17] J. Postel. Transmission control protocol. STD 7, RFC Editor, September 1981. http://www.rfc-editor.org/rfc/rfc793.txt.

[18] Fyodor. Remote os detection via tcp/ip stack fingerprinting. https://nmap.org/nmap-fingerprinting-article.txt. Accessed: 2018-11-26.

[19] T. Stöber, M. Frank, J. Schmitt, and I. Martinovic. Who do you sync you are?: Smartphone fingerprinting via application behaviour. In *Proceedings of the Sixth ACM Conference on Security and Privacy in Wireless and Mobile Networks*, WiSec '13, pages 7–12, New York, NY, USA, 2013. ACM.

[20] G. Taleck. Synscan : Towards complete tcp / ip fingerprinting. 2004.

[21] F. Veysset, O. Courtay, and O. Heen. New tool and technique for remote operating system fingerprinting - full paper -. 2002.

[22] C. V. Wright, L. Ballard, S. E. Coull, F. Monrose, and G. M. Masson. Spot me if you can: Uncovering spoken phrases in encrypted voip conversations. In *2008 IEEE Symposium on Security and Privacy (sp 2008)*, pages 35–49, May 2008.

[23] M. Zalewski. p0f v3: passive fingerprinter. http://lcamtuf.coredump.cx/p0f3/README, 2012. Accessed: 2018-11-20.