Smart Passive OS Fingerprinting using a single TCP SYN Packet

Hidayat ur Rehman
Cyber Security Department
Air University Islamabad, Pakistan
211833@students.au.edu.pk

Nabeel Fakhar
Cyber Security Department
Air University Islamabad, Pakistan
211847@students.au.edu.pk

Muhammad Sangeen
Cyber Security Department
Air University Islamabad, Pakistan
211822@students.au.edu.pk

***Abstract*—The tools available for active OS fingerprinting (such as nmap) send a large number of probes to the target system to identify its OS. If there are no open ports in the target system, it will be nearly impossible to detect its OS by using these techniques. We can overcome this problem by using passive techniques. The tools available for passive OS fingerprinting such as p0f3, Ettercap, Saturi but these tools only analyze the HTTP traffic coming to the host system and it identifies the sender's OS by using TCP SYN-ACK packet. They have a very limited database of signatures and some of these signatures are duplicates of each other. The solution we are proposing is to use a single TCP SYN packet to identify the OS of the target system without disturbing the network traffic. We have generated the OS fingerprints of 11 eleven different signatures including Windows 7, Windows 10, Android 9, 10, 11, LINUX. After testing it, we have come to the conclusion that it is working properly.**

***Index Terms*—OS Fingerprinting, TCP/IP, ARP, Linux**

I. Introduction

Millions of people use computer devices to access bank accounts, purchase goods and services, and keep track of personal information. With the explosive growth of computer networks and usage of internet devices, it has become difficult to maintain the security of these systems. It is very essential to be able to identify a device's OS because it helps to detect and prevent attacks intelligently. One of the most suitable and optimal ways to detect OS is by using TCP SYN packets. [7] There is a lot of information in TCP SYN packet header which, if carefully analyzed, can be used to identify OS. The header information includes MSS (maximum segment size), wsize(window size) etcetera. The fields we are using for the TCP SYN packet header are discussed below.

II. RELATED WORK

TCP/IP fingerprinting has been an active research area. Passive and hybrid schemes are studied as well. Inferring tethering via exploiting different TCP/IP fingerprinting features has been extensively studied. Combining multiple features improves the inferring accuracy. [3] The p0f tool includes five features in TCP/IP header as its signatures in OS detection. [5] The Nmap tool uses a set of nine tests to detect different OS's from network packets. Further optimization techniques to combine multiple features are studied. Our work complements the previous efforts by

1) providing the first comprehensive quantitative study on the effectiveness of passive TCP/IP fingerprinting to OS and tethering detection.
2) Identifying new features for OS fingerprinting and tethering detection
3) Designing a method to effectively combine multiple features.

There are other techniques for detecting OSes or tethering activities, which utilize information in high level protocols, such as application layer features and web browser fingerprints. Unlike TCP/IP fingerprinting, those techniques require Deep Packet Inspection (DPI) [1]. DPI not only has a non-negligible overhead in packet processing, but also raises privacy concerns when adopted by service providers. Besides, the increasing adoption of encryption makes high level protocol information unavailable to use. Due to those practical issues, our study focuses on the features in TCP/IP headers.

A. *Existing Passive OS Fingerprinting Tools*

A few tools already exist that help in detecting operating systems by using passive OS fingerprinting such as

*1) **p0f** :* This tool is an OS Fingerprinting tool that uses a range of sophisticated and purely passive traffic fingerprint mechanisms to identify the players behind any consequent TCP / IP communication (often just a normal SYN) without interfering. [2] It can perform highly scalable and extremely fast identification of the operating system and software at both ends of a TCP connection, especially in environments where NMap probes are stuck, too slow, unreliable, or trigger alarms. The 67-bit signature consists of information about the window size (16 bits), the initial time to live (8 bits), the maximum segment size (16 bits), the "Do not fragment" flag (1 bit), the window scaling option (8 bit), sackOK option (1 bit), nop option (1 bit), initial packet size (16 bit).
P0f utilizes three different detection modes for the purpose:

- SYN mode: Incoming connection fingerprinting – To fingerprint the machine connecting to our system.
- SYN+ACK mode: Outgoing connection (remote) fingerprinting – To fingerprint systems we connect to.
- RST+ mode: Outgoing connection refused (remote) fingerprinting – To fingerprint systems that rejects our traffic.

In its most insignificant mode of operation, p0f only monitors packets that involve the host that is running p0f. This provides a blinkered view of the network. But this might be sufficient if all you want to do is track who connects to your machine.

*2) **Ettercap** :* This tool is a complete suite for man-in-the-middle attacks. It features live connection sniffing, content filtering on the fly, etc. It helps in the active and passive dissection of many protocols. Ettercap is capable of both active and passive dissections of many different protocols. Its ability to perform active eavesdropping and passive OS fingerprinting is what makes it stand out. [6] Ettercap's operating system fingerprint function is not as accurate as p0f. The following information can be obtained by using Ettercap fingerprinting: IP address, operating system, network distance, port, fingerprint, host name, device type. [8] Modes of operations offered by Ettercap:

- IP based: Packet filtering done based on IP (source and destination).
- MAC based: Packet filtering done based on MAC address (source and destination).
- ARP based: Utilizes ARP poisoning (a hacking technique to perform Man-in-the-Middle attack) to sit and sniff traffic on a switched LAN between two hosts.
- Public ARP based: Utilizes ARP poisoning to sniff on a switched LAN from a victim host to all other hosts.

One downside to using Ettercap is the time that has passed since it was last updated. More than four years have passed since its last update was released to users. Ettercap is not compatible with Windows systems or with Solaris and OpenSuSe. Users will need to run Ettercap on Linux or Mac systems for it to work properly. [10] The source compilation of the software requires several dependencies and developer libraries to function properly.

*3) **Satori** :* Satori uses WinPCap. This program listens on the wire for all traffic and does OS Identification based on what it sees. It currently supports fingerprinting by the following means:

- DHCP
- TCP
- HTTP (User Agent and Server)
- SMB (TCP and UDP)

As the result of OS fingerprinting, Satori offers IP address, fingerprint and a list of possible OSs with their weight up to 12. The higher the weight, the bigger the possibility.

*4) **Network Miner** :* Network Miner is a network forensic analysis tool for Windows. Network Miner acts as a passive packet sniffer and captures data over the network, without generating new traffic on its own and thus remains secret. It uses the WinPcap library in order to capture packets and it can also scan PCAP files offline.

III. Methodology and Objectives

There are two common methods for performing fingerprinting: active and passive scanning.

Active: - OS fingerprinting requires the use of a set of specialized probes that are sent to the system(target) and the system responses from this active probing give information about what type of OS might be installed. [9]

Passive: - No traffic is sent to the target system with passive fingerprinting. Passive OS fingerprinting involves sniffing network traffic at any given collection point and matching that pattern with pre-established OS patterns/identities.

One of the aspects of active methods is that the probing traffic can be malicious. This might harm or crash the target system. The packets can also be easily detected by IDS (Intrusion Detection System). This might lead to negative consequences such as complaints against research institutions using these methods and possibly reduced accuracy of the results. Moreover, in passive technique we maintain a database for the purpose of identification, but unlike the active technique, it does not generate any traffic. It simply sits and sniffs the packets sent by the target system and based on the unique signature of the operating system in the packet, it determines the OS. As no new packets are generated and sent to the target machine, it won't trigger any security measures put in place by the owner.

IV. Proposed Research Solutions

The paper is proposing two different solutions. One is to suggest a passive technique for detecting the OS of devices connected to the network and the other is for generating TCP SYN signatures of the OS. We will apply ARP poisoning on all the devices connected to the network one by one until we capture the TCP SYN packet of the target device and we block the target device for a maximum of two seconds. If we are able to get the TCP SYN packet before 2 seconds, then we end the ARP attack. All the tests conducted so far have proved successful. 95% of the time, we were able to get the TCP SYN packet of the device before 1 second if the system is online. In this way, the user's traffic flow remains smooth as the captured packet is retransmitted to the router due to TCP's default behavior. If any device does not send the packet within 2 seconds, we temporarily store the device and after every passing minute, we launch ARP poisoning attack on the selected IP for 2 seconds until the system is running.

*A. **Dataset Gathering technique***

We have gathered the signatures' data of various OS by using our website. The website was deployed on the server. It will store the IP, accessing time, and user agent. User agent contains the user's system OS, and browser name. The users accessing the website. A packet sniffer was deployed on the server. The server captured the TCP SYN packet, time of receiving, and IP of the sender. Then we mapped both tables and successfully stored the OS and Its signatures of different users accessing the website.

*B. **OS FINGERPRINTING***

We are using the p0f format of generating signatures with some modifications. The signature format is as follows [4]
sig = ver:ittl:olen:mss:wsize,scale:olayout:quirks:pclass

ver - signature for IPv4 ('4'), IPv6 ('6'), or both ('*').
ittl - initial TTL used by the OS. Almost all operating systems use 64, 128, or 255; ancient versions of Windows sometimes used 32, and several obscure systems sometimes resort to odd values such as 60.

olen - length of IPv4 options or IPv6 extension headers. Usually zero for normal IPv4 traffic; always zero for IPv6 due to the limitations of libpcap.

mss - maximum segment size, if specified in TCP options. Special value of '*' can be used to denote that MSS varies depending on the parameters of sender's network link, and should not be a part of the signature. In this case, MSS will be used to guess the type of network hookup according to the [mtu] rules.

wsize - window size. Can be expressed as a fixed value, but many operating systems set it to a multiple of MSS or MTU, or a multiple of some random integer. P0f automatically detects these cases, and allows notation such as 'mss*4', 'mtu*4',
scale - window scaling factor, if specified in TCP options. Fixed value or '*'.

olayout - comma-delimited layout and ordering of TCP options, if any. This is one of the most valuable TCP fingerprinting signals. Supported values:
eol+n - explicit end of options, followed by n bytes of padding
nop - no-op option
mss - maximum segment size
ws - window scaling
sok - selective ACK permitted
sack - selective ACK (should not be seen)
ts - timestamp
?n - unknown option ID n

quirks - comma-delimited properties and quirks observed in IP or TCP headers:
df - "don't fragment" set (probably PMTUD); ignored for IPv6
id+ - DF set but IPID non-zero; ignored for IPv6
id- - DF not set but IPID is zero; ignored for IPv6
ecn - explicit congestion notification support
0+ - "must be zero" field not zero; ignored for IPv6
flow - non-zero IPv6 flow ID; ignored for IPv4
seq- - sequence number is zero
ack+ - ACK number is non-zero, but ACK flag not set
ack- - ACK number is zero, but ACK flag set
uptr+ - URG pointer is non-zero, but URG flag not set
urgf+ - URG flag used
pushf+ - PUSH flag used
ts1- - own timestamp specified as zero
ts2+ - non-zero peer timestamp on initial SYN
opt+ - trailing non-zero data in options segment
exws - excessive window scaling factor (¿ 14)
bad - malformed TCP options If a signature scoped to both IPv4 and IPv6 contains quirks valid for just one of these protocols, such quirks will be ignored for on packets using the other protocol. For example, any combination of 'df', 'id+', and 'id-' is always matched by any IPv6 packet.
pclass - payload size classification: '0' for zero, '+' for non-zero, '*' for any. The packets we fingerprint right now normally have no payloads, but some corner cases exist.

V. Contributions

Given the many open issues in wide-scale fingerprinting and lacking performance analysis in the literature, we will formalize the estimation problem in single-packet OS classification and analyze the drawbacks of existing techniques. We then develop our own technique for OS detection and efficient data collection.

VI. CONCLUSION

This paper develops and evaluates a methodology that uses several features in network traffic for identifying the OS on the sending device. This OS fingerprinting can be used for detecting tethering and more generally deployment of network address translation. The proposed methodology includes a probabilistic approach to combine multiple features to enhance detection accuracy. We evaluate the effectiveness of individual features and find TTL, TCP timestamp, and TCP window size scale factors are more accurate, while clock frequency and boot time are less accurate. Furthermore, we proposed a solution for generating dataset for signature effectively.

VII. Future Work

Our system cannot detect the OS of those systems whose signature is not stored in the database. We can use AI techniques for generalizing our existing signature so that we can use it for those devices. Also, fuzzy searching can be enabled. This includes, for example, capabilities for searching similar requests on the base of a fingerprint's substring, using a wildcard search, or searching depending on the importance of fingerprint elements. Another direction is to use our system as a basis for request clustering mechanisms, which can help to uncover new relations between requests.

References

[1] Shamsi, Z., Nandwani, A., Leonard, D., Loguinov, D. (2014). Hershel: single-packet os fingerprinting. ACM SIGMETRICS Performance Evaluation Review, 42(1), 195-206.
[2] Montigny-Leboeuf, D. (2005). A multi-packet signature approach to passive operating system detection. DEFENCE RESEARCH AND DEVELOPMENT CANADAOTTAWA (ONTARIO).
[3] Rights, R. F. (2003). Global information assurance certification paper. GIAC.
[4] Zalewski, M. (2012). p0f v3 Passive fingerprinter. GitHub. https://github.com/p0f/p0f
[5] Chen, Y. C., Liao, Y., Baldi, M., Lee, S. J., Qiu, L. (2014, November). Os fingerprinting and tethering detection in mobile networks. In Proceedings of the 2014 Conference on Internet Measurement Conference (pp. 173-180).

[6] Białczak, P., Mazurczyk, W. (2021). Hfinger: Malware HTTP Request Fingerprinting. Entropy, 23(5), 507.

[7] Medeiros, J. P. S., Júnior, A. D. M. B., Pires, P. S. M. (2011). A qualitative survey of active TCP/IP fingerprinting tools and techniques for operating systems identification. In Computational Intelligence in Security for Information Systems (pp. 68-75). Springer, Berlin, Heidelberg.

[8] Patil, R. Y., Devane, S. R. (2019). Network Forensic Investigation Protocol to Identify True Origin of Cyber Crime. Journal of King Saud University-Computer and Information Sciences.

[9] Husák, M., Čermák, M., Jirsík, T., Čeleda, P. (2016). HTTPS traffic analysis and client identification using passive SSL/TLS fingerprinting. EURASIP Journal on Information Security, 2016(1), 1-14.

[10] Ulery, B. T., Hicklin, R. A., Buscaglia, J., Roberts, M. A. (2011). Accuracy and reliability of forensic latent fingerprint decisions. Proceedings of the National Academy of Sciences, 108(19), 7733-7738.