future internet

MDPI

Article

SDMw: Secure Dynamic Middleware for Defeating Port and OS Scanning

Dalal Hanna, Prakash Veeraraghavan * and Ben Soh

Department of Computer Science and Information Technology, La Trobe University, Victoria 3086, Australia; D.Hanna@latrobe.edu.au (D.H.); B.Soh@latrobe.edu.au (B.S.)

* Correspondence: P.Veera@latrobe.edu.au, Tel.: +61-3-9479-1547

Received: 27 September 2017; Accepted: 18 October 2017; Published: 21 October 2017

Abstract: Fingerprinting is a process of identifying the remote network devices and services running on the devices, including operating systems (OS) of the devices, and hosts running different OSs. Several research proposals and commercial products are available in the market to defeat fingerprinting. However, they have performance limitations and expose themselves to attackers. In this paper, we utilize some real-time fault-tolerance concepts (viz. real-time/dynamic, detection/locating, confinement/localizing and masking/decoy) to propose a plug-and-play adaptive middleware architecture called Secure Dynamic Middleware (SDMw) with a view to defeat attackers fingerprinting the network, without exposing itself to the attackers. We verify that the proposed scheme works seamlessly and requires zero-configuration at the client side.

Keywords: port scanning; OS-fingerprinting; network scanning; dynamic middleware

1. Introduction

The Internet has become all too pervasive in our modern society. It has come to dominate our lives as never before. Every day, more bits of data flow across the Internet than there are grains of sands on all the beaches in the world. According to *Cisco* [1], the total global Internet traffic for 2011 was approximately 8.4×10^{18} bits per day. This excludes intranet traffic. Meanwhile, University of Hawaii estimated that the number of grains of sands on all beaches in the world be approximated at 7.5×10^{18} grains [2]. Furthermore, *Cisco* predicted that there is an annual increase of 32% of the Internet Protocol (IP) traffic every year. When it comes to the network topology, new domains are added to the Internet almost every day. Thus, the Internet has passed the point where it is hard to visualize, comprehend or control the activities across the networks. Since the Internet connects politically diversified regions, the law and regulations in one region may not impact or mean anything in other regions. Consequently, protecting the network and privacy of the user data is the foremost important research priority today. A significant challenge for today's network engineers and security administrators is to develop techniques capable of detecting attempts to compromise the integrity and availability of the network.

The rest of the paper is organized as follows. In Section 2, we provide background relating to our work. In Section 3, we provide a review of operating systems (OS) fingerprinting.

Section 4 deals with port-scanning techniques. OS fingerprinting and port scanning are forerunners for any attackers intruding a network. These two sections provide a strong motivation for our proposal. In Section 5, we propose our Secure Dynamic Middleware (SDMw) architecture. To validate its practicality, we analyse the performance of the proposed architecture, in terms of introduced latency. Section 6 deals with conclusions and future directions.

2. Background and Aims

The Internet uses the Transmission Control Protocol and the Internet Protocol (TCP/IP) suite of protocols that were designed with no security in mind. Both the research community and developers of this protocol did not anticipate this exponential growth. Thus, these protocols offer minimal protection against eavesdropping. Today, the Internet has become part of our day-to-day life. We use the Internet to shop, book holidays, file tax returns or work on a complex collaborative project. A wealth of corporate and personal information is stored online and is accessible through the Internet. Breach of privacy and confidentiality of data may cause huge financial loss and possibly human loss.

Network security is essentially a battle between a Penetrator who tries to find holes and the Administrator who tries to close them before it is being found. Security is only as strong as the weakest link. The attacker often uses the weakness found in the protocol, hardware, or the combination of both to penetrate the network.

Fingerprinting is a process of identifying the remote network devices, services running on the devices, their Operating system (OS), hosts running different OSs and so on. Fingerprinting a network equipment has several potential applications and benefits in network management and security. It is also useful to understand the network structures and their behaviour. However, to penetrate a corporate network, it is essential to fingerprint their network topology, the software system and the services used in the network. Fingerprinting is done either from a single remote computer connected to the Internet or from a group of computers. The attackers often collaborate from different parts of the world to launch an attack against a network. The attackers use either a spoofed IP address or some compromised proxy servers to hide their identity. To launch a distributed scanning, the attacker may install zombies and control them from a central remote computer.

Several research proposals and commercial products are available in the market to defeat fingerprinting. Some of them have performance limitations, whereas others expose themselves to the attacker. In this paper, we propose a plug-and-play adaptive middleware architecture to defeat attackers fingerprinting a network, without exposing itself to the attackers. We also demonstrate that our proposed scheme works seamlessly and requires zero-configuration at the client side.

3. Network and Host Fingerprinting: An Attacker Perspective

A precursor to a *port scanning attack* is network and host fingerprinting. Our proposed architecture will focus on these two elements: fingerprinting and port scanning. We will review port scanning in Section 4, and, in this section, we review the state-of-the-art in network and host fingerprinting from the attacker's viewpoint. Network security is a critical element in any organization; however, security has become one of the primary concerns of computer professionals and organizations today. The aim of network security is to protect private data and also to ensure that the network resource and Internet connections are not being compromised and misused.

To intrude into any network, as the first task, an attacker needs to fingerprint the network. As human fingerprints uniquely identify a person, we can identify an Operating System on a network through specially crafted TCP and IP packets (called packet fingerprinting) with specific parameters. The TCP/IP specification has several ambiguities dealing with initial parameters. Different OS vendors interpret these ambiguities in a different way. By analysing initial values of protocol flags, options and packet fields, we can determine the OS installed on a host. Based on the OS, we can also discover the hardware (like switches and routers). Even though fingerprinting a network is not seen as an attack, they often serve as a prelude to further attacks by exploiting known bugs in the remote host's OS.

Fingerprinting is done either actively or passively. Active fingerprinting involves sending some specially crafted TCP/IP packets to the target network and observing their reply. The replies are then matched against known Operating Systems and Services. A malicious use of the fingerprinting technique is to construct a database of a noted IP address and the corresponding Operating Systems

for an entire network. When someone discovers a new exploit, the attacker can target only those machines running the vulnerable OS. Almost every system connected to the Internet is vulnerable to fingerprinting, including webcams, fax machines, printers and so on.

The TCP/IP protocol was designed in the 1970s and has undergone several changes. However, it still has several design ambiguities. Both of these protocols are described in their respective Request For Comments (RFCs), (791 and 793 for IP and TCP, respectively) for developers to read and understand when implementing. However, there are areas where the RFCs leave certain decisions to the developer. The protocol specification did not specify how certain modules need to be implemented, especially when dealing with initial parameters and the use of specific flow control algorithms.

1 Time-to-Live (TTL) Value: Several OSs set different initial TTL values. For example, Windows 10 (Microsoft Corporation, Redmond, WA, U.S.A) uses a default value of 128; Linux use 64 and Solaris (Oracle Solaris, Redwood Shores, CA, U.S.A) use 254.

2 TCP-Initial Sequence Number (ISN): Even though almost every OS uses non-guessable ISN (after Kevin Mitnik's famous attack), the microscopic drift in the Central processing unit (CPU) clock cycle is used in the literature to enumerate the number of hosts inside a Network Address Translated (NATed) network.

   Even though every host behind a corporate network uses Network Time protocol (NTP) to synchronize their clocks, there is always a microscopic drift in their clock. This is due to inaccuracy in the clock crystals. The TCP-ISN generation algorithm uses the system clock information for the randomization process. For a host outside a NATed network, every connection appears to originate from a single IP (or a small pool of IP) source. However, the microscopic drift in the individual computer's Realtime clock inside a NATed network is exploited to count the number hosts behind a NATed network.

3 There are few TCP options that will not reveal the OS information by itself. However, it can be used along with other parameters to narrow down the OS detection. They are TCP-timestamp option, Window Scaling, Maximum Segment Size (MSS), and Explicit Congestion Notification (ECN).

4 IP-Identification field (IP-ID): The IP-ID field is a 16-bit field in the IP header. IP packets can also be tagged with a Don't Fragment (DF) bit, in which case the ID field can be ignored. Since the source host has no knowledge about the path-MTU (maximum transmission unit), every IP packet contains a unique IP-ID value. Windows 95, 98 and NT family increment the IP-ID field by 128, rather than 1. From Windows 2000, it was set to 1 like any other OS. Currently, IP-ID did not reveal much of information about OS; however, different IP-ID mappings are used to enumerate the number of hosts inside a NATed environment. When replying to an ICMP-Echo message, Linux kernel use an IP-ID value of 0 and the Don't Fragment (DF) Flag set to 1. This makes it easy for the attacker to identify Linux kernel.

5 If a TCP-synchronize (TCP-SYN) or a TCP-Finish (TCP-FIN) segment is sent to a closed port, different OS running on a host will behave differently. This will facilitate OS detection.

Traditionally, Network Address Translation (NAT) allows multiple hosts in a private network (using private IP addresses) to share either a pool of Public IP addresses or a single public IP address (in this case, the scheme is called a Port Address Translation (PAT)). Originally, NAT was designed to be a short-term solution to the problem of IPv4 address depletion. Today, it is used for several different purposes, including the protection of the corporate network. A NAT deployment ranges from private home users with few machines to large corporate networks behind NAT-gateways. NAT operates by changing only the source IP address and it does not modify any higher layer protocol information such as the Transmision Control Protocol (TCP) or the User Datagram Protocol (UDP). Since private IP addresses are non- routable, they cannot be used as a destination address in the Internet. Thus, an attacker may not be able to fingerprint them. Because of this unique feature, the use of NAT is widely preferred for hiding a corporate network through the use of private IP

addresses. Obtaining information about hosts behind a NATed network is an interesting research area. Several research articles are available in the literature on counting the number of hosts behind a NATed network.

There are several commercial and free-to-use tools available for fingerprinting OS. However, the most complete and the widely used OS fingerprinting tool is the Network Mapper (NMAP) [3]. It uses a database of over 500 fingerprints to match various OS and hardware platforms. Fingerprints for new hardware and OS are added everyday through the online community [3]. The NMAP system identifies the target OS and hardware by sending up to fifteen TCP, UDP and Internet Control Message Protocol (ICMP) probes. These probes are specially crafted packets designed to exploit various ambiguities in the standard RFC implementation of the protocol. NMAP compares responses with the known pattern for finding out the remote OS.

In [4], Khono et.al. used the microscopic drift in the clock-skew to fingerprint remote devices, rather than OS. Using their proposed method, an attacker can remotely probe a block of address to determine if it corresponds to virtual hosts. We note that the block of addresses must be in a public IP range. In addition, to improve the accuracy, Khono's algorithm needs long-term data. Thus, it is efficient to use tools like NMAP at first to narrow down the search before launching Khono's algorithm.

In a practical environment, it is not possible to use several packets for fingerprinting a host (or OS). In a recent paper, Shamsi et al. [5] proposed a single packet OS fingerprinting. As mentioned in their work, the current fingerprinting tools like NMAP are not well suited in the Internet environment due to large amount of traffic generated, and the intrusive nature of the probes. They presented a proposal to fingerprint an OS using a single TCP-SYN frame. They also build a database consisting of 116 Operating Systems. However, the downside is that new firewalls issue a TCP-Reset (TCP-RST) segment for any incoming connection to hosts that do not run any registered TCP services. Thus, in an unknown environment, Shamsi's proposed method may trigger several false positives due to firewalls issuing TCP-RST segments.

The above-mentioned methods use active fingerprinting. Using the above methods, an attacker can fingerprint any host that has a public IP address. Passive fingerprinting systems simply sniff the network traffic and opportunistically classify the hosts as they observe their traffic. This is more time-consuming and difficult to comprehend compared to active fingerprinting. Most of the time, the traffic sniffer may not be on the active-path between the source and the destination host to sniff their traffic. However, it is hard to detect the presence of a passive system in the network. Fingerprinting works well in distinguishing different types of known Operating Systems or Network Services, but it may not detect different versions of the same operating system or network service.

4. Port Scanning

Once the fingerprinting is complete, an attacker will construct a database of IP addresses and the corresponding Operating Systems for an entire network. When someone discovers a new exploit, the attacker may target only those machines running the vulnerable OS. The next step is to perform a port scanning. It is the process of probing a host for open ports and other services running on the host. Several TCP and UDP services listen at various ports for possible incoming connections. For example, The Hypertext Transfer Protocol (HTTP) servers listen for connection at port 80. Some ports may easily be exploitable compared to other ports. Services listening on these ports may allow under privileged connections to manipulate system files or install additional services. Port scanning is a method for discovering exploitable communication channels.

Port scanning is divided into two main parts: Horizontal and Vertical scanning. In a horizontal scanning, the same port (number) is scanned over multiple hosts in the same network. This type of scanning is useful for attackers who want to gain control of victim hosts by exploiting a known vulnerability of a certain service running on that port. In a vertical scanning, multiple ports are scanned on the same host machine. This technique is common for an attacker who is gathering information

to fingerprint a host machine and to attack at a later point of time, for example, Web servers, The Domain Name System (DNS), Dynamic Host Configuration Protocol (DHCP) servers and so on.

There are several techniques available for surveying the open and closed TCP/UDP ports on a target machine. Each technique has its own strength and weaknesses. The most common ones are as follows: [6]

(1) TCP-connect(): This is a traditional way of making a TCP system call to establish a connection to an interesting port in a target host. If the destination host is listening on a particular port, the `connect()` request will succeed; otherwise, the port is unreachable. This method has the following two best advantages compared with other methods: the first advantage is that it does not require any administrative privilege to launch this request. The second one is the speed. The attacker can hasten the scan using many sockets in parallel. However, the downside is that this type of scan can be easily detected and can be filtered.

(2) TCP SYN scanning: the attacker sends a TCP-SYN packet as if he is going to open a "real" connection and waits for the SYN-Acknowledgement (SYN-ACK). A SYN-ACK indicates that the port is open and listening. A TCP-Reset (TCP-RST) indicates that the port is either not available or not willing to accept a connection at this point of time. Since we are trying to establish a "Simplex" connection between the attacker and the target host, this technique is often referred to as "half-open" scanning. One of the disadvantages is that newer firewalls immediately issue a TCP-RST segment without even referring the packet to the destination host.

(3) TCP-FIN Scanning: This method is used to overcome the disadvantage of the TCP-SYN scanning. Newer firewalls and packet-filters watch for any TCP-SYN segment sent to a prohibited port (i.e. ports that are not open). Once a series of TCP-SYN is detected, they block the attacker's IP address either permanently, or for a timeout period. However, if an attacker transmit a TCP-Finish (TCP-FIN) segment to a closed port, the host tends to reply to the TCP-FIN segment with a proper TCP-RST. TCP-FIN segments may pass through the network without getting detected.

There are several other methods available in the literature like Fragmentation attack, Reverse Indent scanning, ICMP-ping etc. However, they are either less frequently used or patched by the latest firewalls. There are a number of free and commercial tools available that perform detailed port scanning. In contrast, NMAP is the most powerful scanning tool available for free-to-use [3]. NMAP can also be used for a number of other purposes. Even though there are a number of tools ready for the use of performing port scanning, there are very few tools available for detecting port scanning attempts.

Most of the solutions, including Antivirus, Firewalls, and an Intrusion Detection System (IDS), can indicate an occurrence of a port-scanning attack or an unauthorized port activity. As port scans are usually performed before an actual attack, indications of port scan activity give up a clue that an attack might follow in the future. Thus, having an intelligent system that predicts the occurrence of attacks soon is preferable.

In the recent past, port-scanning detection has received a lot of attention by researchers. Several intelligent algorithms using Artificial Intelligence-techniques were invented to detect possible port scanning attacks. Intrusion-Detection Systems use these algorithms to block port scanning attempts. For a detailed review on various port scanning techniques, one may refer to [7]. This book is an excellent resource of network scanning. In [8], the authors have used data mining techniques for network intrusion detection. They built a class of prediction-model for identifying known intrusions and their variations, anomaly/outlier detection schemes for detecting attacks whose nature is unknown. They validated their model using experimental results on the DARPA-1998 data set, as well as on live network traffic at the University of Minnesota. However, the validity of their model for detecting current intrusions need to be evaluated.

In [9], Soniya and Wisey used a Neural network approach to detect TCP-SYN scanning using packet counts. They trained a Neural Network to capture the behaviour for normal as well as port scan data. Their results show the normal and abnormal behaviour of TCP-SYN, SYN-ACK and TCP-FIN packet sequence. A deviation from the normal behaviour is used to effectively detect TCP-SYN scanning without maintaining a state information.

In [10], Baig and Karman used a time independent feature set to detect port scans. Their model can detect slow and random attacks in real time with reduced memory needs. Their proposed model was tested using DARPA-99 data set. In [11], Leckie and Kotagiri used a probabilistic approach to detect port scans. Their model considered both the number of destinations and the ports accessed by an attacker, in order to determine how unusual the access is.

However, in all the above research work, or in commercial IDS systems, the authors and the developers assumed that the attackers scan the network in a short duration. That is, the interarrival rates between the successive scan packets are small. Even though this assumption was valid until few years back, it is not valid any more. An attacker may use either a proxy chain to launch a scanning or deploy zombies to launch distributed scanning. In both cases, every scan packet arrives from a different IP address. Thus, blocking a single IP address is not effective anymore. Nowadays, attackers use a combination of zombies along with proxy-chains to launch a more powerful attack, which is hard to detect.

5. SDMw: The Secure Dynamic Middleware Architecture

Due to increase in OS vulnerabilities, enterprise network administrators regularly perform OS audits. They must also audit for any open unauthorised ports running underprivileged services. The audits also help them in maintaining enterprise network policies. Network administrators use tools such as NMAP to perform this audit. Thus, OS fingerprinting and port-scanning are considered a healthy way to maintain enterprise security policies. However, the hacking community use this mechanism for their advantages. If an enterprise deploys a commercial tool to defeat fingerprinting, then the tool itself will exhibit its behaviour to the attackers. In that case, an attacker may launch a distributed denial-of-service (DDoS) attack against the tool. Rather than defeating a fingerprint or port scan process, in this paper, we propose the SDMw architecture for sending fictitious information about the network to the attacker.

SDMw is a plug-and-play adaptive middleware that will exhibit different OS characteristics at different times, with a view of defeating attackers fingering the network in a transparent manner.

5.1. The SDMw Architecture

As a first step in our research proposal, we collect several popular OS fingerprints. The SDMw system uses each specific OS fingerprint at different points of time.

5.1.1. The Unified Header Generation Algorithm (UHGA)

We consider several popular OSs like Windows, Linux, Solaris, etc. As a first step, we adopt a unified algorithm to determine the TCP/IP parameters (including IP-ID, TCP-Initial Sequence Number (TCP-ISN), timestamp, the choice of a congestion control algorithm, window size, etc.) that are typical to the OS we considered. Let every host in the network use the same unified algorithm transparently. For an attacker, every host (including legacy hardware devices like printers, fax etc.) appears to run the same OS, (either known OS, in case we adapt known OS fingerprint parameters; or unknown OS if we design our own algorithm to decide on the TCP/IP parameters).

5.1.2. SDMw Placement

Ideally, SDMw is placed between a corporate network and the external world. Due to its light weight, it can run as a proxy service in any subnet or it can run as a proxy middleware in every host. However, placing the SDMw system as a proxy service in every subnet is the most appropriate

placement. If the SDMw service runs on every host, then the network administrator will not be able to fingerprint a host OS for legitimate audit purposes. In addition, updating the SDMw system in every host consumes time and is unrealistic.

5.1.3. The SDMw Black Box

The black box of the proposed system is shown in Figure 1. The core engine consists of a pluggable module that determines different versions of OS TCP/IP-kernel parameters at different periods of time. This implements the UHGA. The in–out table is a translational table, similar to the one kept in a Socks-proxy or a TCP-intercept system. For every pair of a source and a destination host, depending on the time, the SDMw system intercepts the connection and generates TCP/IP header parameters that appear to represent a fingerprint of a particular OS at that time. For different times, SDMw will use different OS TCP/IP parameters. By doing this way, either the OS detection by remote attacker is masked or provides ambiguous results.

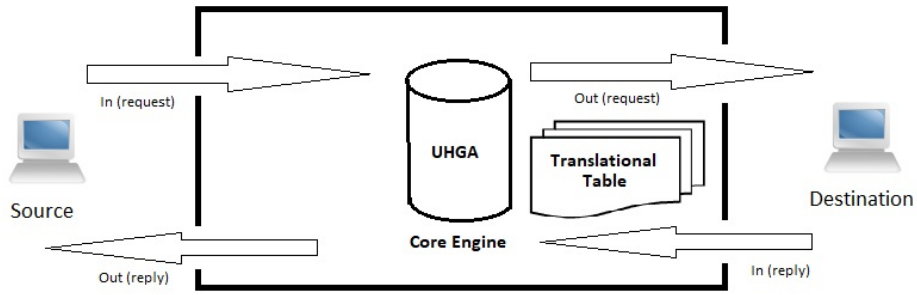

Figure 1. The Secure Dynamic Middleware black box.

We now discuss how SDMw handles connectionless and connection-oriented services originating from inside hosts.

5.1.4. Connectionless Datagram Service from Inside Hosts

UDP and IP datagrams offer connectionless services. The term connectionless means that UDP and IP do not maintain any state information about successive datagrams. Each datagram is handled independently from all other datagrams. For connectionless UDP services, source IP address, source port, destination IP address and the destination ports (called a socket) information are important. Each UDP connection is uniquely identified by this 4-tuple. An attacker may not be able to fingerprint a system just based on this parameter. Thus, the SDMw system preserve this 4-tuple without modification. If the source IP address and the source port information is changed (as in the case of a Network Address Translation (NAT) or a Port Address Translation (PAT) system), a reference translational table must be maintained in order for the reply to be forwarded to the right host. If a NAT/PAT is not used along with SDMw, the proposed SDMw system will not modify the original 4-tuple. Thus, it will improve the system performance by reducing a table-lookup latency.

At Layer-3, IP-ID field, TTL, Explicit Congestion Notification (ECN) and Fragment flags are traditionally used for fingerprinting a host. These parameters are chosen by the host's OS on per packet basis, and they need to be changed in order for a host to be protected from fingerprinting. If we assume that IP fragmentation is not needed along the path between a host and the SDMw, then changing them by the SDMw is not going to affect the communication between the end-hosts. Since these parameters are applied on per packet basis, there is no need for the SDMw to maintain a translational table. Our proposed SDMw architecture will generate these parameters according to the chosen UHGA algorithm and replace the one in the IP packet.

5.1.5. Connection-Oriented TCP Service from Inside Hosts

Even though TCP and UDP uses the same network layer (IP), TCP provides a totally different service to the application layer than UDP does. TCP provides a connection-oriented, reliable, byte stream service. Like UDP services, TCP connections are also identified by the 4-tuple, source IP address, source port, destination IP address and the destination ports. The term connection-oriented means the two applications using TCP (normally considered a client and a server) must establish a TCP connection with each other before they can exchange data. The logical connection is established and maintained by means of 32-bit sequence and acknowledgement numbers. This process starts with an Initial Sequence Number (ISN). TCP is very sensitive on establishing, maintaining and tearing a connection. An on-going connection will be terminated if there is a mismatch in the sequence number.

As before, from a socket, no host OS information is exposed to the attacker. However, compared to IP and UDP, TCP has several design ambiguities that are traditionally exploited to fingerprint a host. Our proposed SDMw architecture preserves the original socket information. However, we change other TCP parameters, including the sequence numbers that can mask the OS detection. Since we are changing the logical connection information, we need to maintain a translation table in order for the reply from the destination to be forwarded to the end-host properly without tearing an on-going connection. While deciding on the congestion window parameters, SDMw chooses the minimum of the one it generates based on the UHGA and the host's advertised congestion window parameters. Based on the particular OS chosen by the UHGA algorithm, the TCP parameters are determined on a per-socket basis. Entries in the translation table are maintained for $2 \times MSL$ (maximum segment lifetime) wait period before they are removed.

5.1.6. Connection from External Hosts to an Internal Host

Any external host may initiate either a connectionless, or a connection-oriented service to an inside host. Since SDMw is positioned between the external and internal hosts, like a firewall, any such connection must pass through SDMw. If an external host is initiating a connection to an unregistered port of a host, rather than the end-host handling this message, SDMw handles on behalf of the end-host. Traditionally, attackers fingerprint a remote host by sending probe packets to closed, as well as open ports. They also do port scanning in order to launch an attack. To protect the end host from fingerprinting and attacking at a later point of time, SDMw replies on behalf of end-hosts. Depending on the particular OS fingerprint chosen by the UHGA algorithm, the proposed SDMw architecture will handle the request as if the particular OS is running on the end-host. If an end-host is running a registered service, then SDMw will forward the connection request to the end-host. Thus, it is necessary to keep all the registered service information with the proposed SDMw system.

If an external host initiates a connectionless IP service such as ICMP, the IP packet will be dropped by SDMw without referring to the end-host. SDMw follows in a similar fashion if either an UDP or a TCP connection is initiated to an unregistered port. SDMw will allow only connection to a publicly-registered port (like HTTP ports).

Whenever an internal host requests a connection (or sends an IP packet) destined to an external Host B, the packet is received by the ingress link of the black box. Based on the source, destination and the current time, the UHGA algorithm will generate and replace the original TCP/IP header fields. Other application layer fields are kept intact. The source information, including the original TCP/IP fields are copied to the same row as in the incoming packet at the Egress-column and the packet is transmitted through the outside (egress) link. Once the reply packet is received, the reverse of ingress–egress entry is used to relay the reply to the inside host. This process is similar to a Socks-proxy server. However, the main difference is that the SDMw system rewrites the entire protocol header fields to pretend as if the machine is using a certain OS.

For connectionless UDP and IP packets, it is not necessary for the SDMw system to use the in–out table. However, for TCP connection, the system must maintain the in–out table for forwarding

purposes. Figure 2 depicts the operation of the SDMw system when using a TCP connection. This process is similar to TCP-intercept or a Socks proxy server.

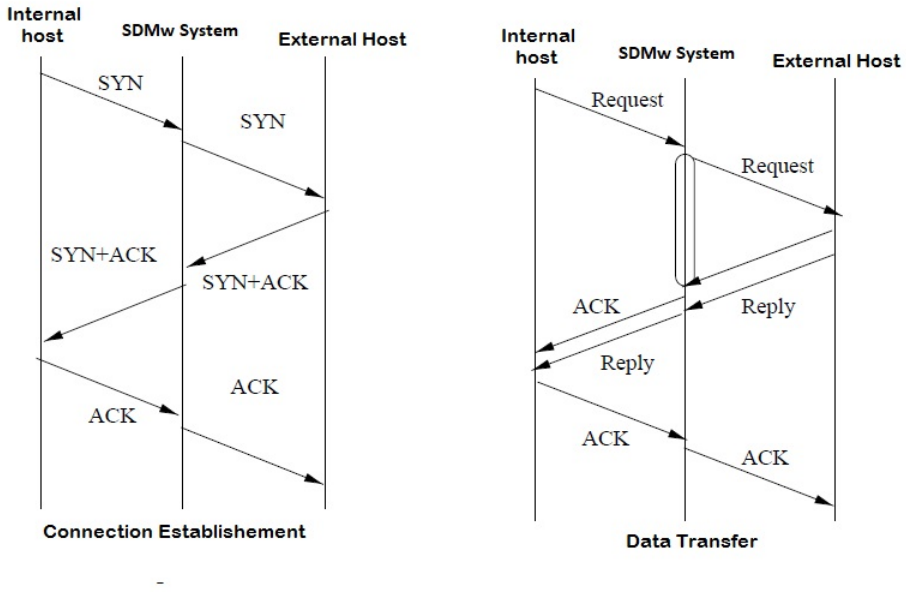

Figure 2. Transmission Control Protocol connection establishment.

In communication, deadlock occurs when Process *A* is trying to send a message to Process *B*, which is trying to send a message to Process *C*, which is trying to send a message to Process *A*. This type of deadlock must be handled by the Operating System. Discussing system and hardware deadlock is beyond the scope of this paper. In the area of distributed algorithms, a deadlock may arise between two algorithms *A* and *B*, when Algorithm *A* is expecting an input from Algorithm *B* and Algorithm *B* is expecting an input from Algorithm *A* in order for them to proceed to the next state.

However, from the state-transition diagram of the TCP/IP protocol [12], the deadlock due to the TCP/IP algorithm is not possible. The protocol specification has a clearly defined set of timing and procedures to either drop a packet/frame or to abort a connection in the event of any mismatch in the expected parameters or a long wait.

In a proxy-middleware system, deadlock occurs, whenever a reply from a destination host cannot be forwarded to the correct host; i.e., the destination address in a received unicast packet matches those of several inside hosts (due to multiple matches in a translation table). This may be due to a poorly implemented software system or not following the IETF specifications.

It is easy to establish the following result.

Theorem 1. *The proposed SDMw architecture works without any deadlock if and only if the underlying UHGA produces unique outputs without deadlock.*

Our proposed SDMw architecture only changes certain fields in the TCP/IP header (according to some established OS header generation scheme). We keep the source and the destination address intact. Since the underlying UHGA uses known OS header generation algorithms, it will not result in potential deadlock.

For connectionless ICMP and IP packets, the proposed SDMw system keeps the source and the destination IP address intact. SDMw only changes the source IP-ID and flags based on the current OS fingerprint in use. The potential reply from a destination host solely depends on the Source IP address, and thus cannot cause any possible deadlock. Even if two pairs of <source IP

address, destination IP address> use the same set of parameters, it is not going to cause any potential deadlock.

For a connection-oriented TCP service, SDMw keeps the socket information unchanged. The proposed SDMw changes the TCP frame information like ISN, MSS, timestamp and other OS specific fields. Changing these fields are not going to cause any potential deadlock. However, to maintain an orderly TCP communication, SDMw maintains a translational table that keeps track of the changes made to sensitive TCP parameters like ISN, on-going sequence and ACK numbers. Since the socket information is kept intact, the TCP algorithm guarantees a deadlock-free operation.

5.2. The Performance of the SDMw System

In this section, we evaluate the performance of the proposed SDMw system in terms of the latency. We implemented a simple test-bench topology that consists of two hosts running Windows OS. They are connected to a switch, which is then connected to the SDMw system. The striped-down prototype of the proposed SDMw is implemented in a Linux box with two network cards. One network card (say CARD1) is connected to the switch and the other network card (CARD2) is connected with a server machine running the HTTP service. The SDMw is implemented as a proxy-intercept service between these two network cards. The SDMw uses the native Linux TCP/IP kernel behavior. The prototype architecture is presented in Figure 3. We tested a connectionless ICMP and a connection-oriented TCP service from Host 1 and Host 2 to the server. Whenever an IP packet is received at CARD1, we record the current system time.

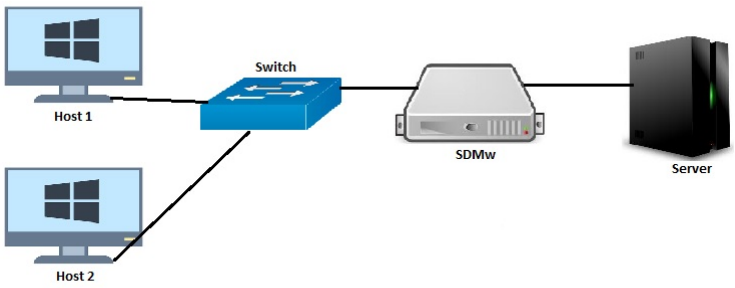

Figure 3. Secure Dynamic Middleware (SDMw) test bench topology.

1. If the protocol number in the IP packet is 1 (ICMP), we keep the source and the destination IP address unchanged. We change IP-ID field and flags to reflect the underlying TCP/IP kernel behavior (as SDMw is using the native Linux kernel behavior). We then record the time in which the packet was forwarded to CARD2. The difference between the two timestamps gives us the latency due to SDMw processing, which includes OS scheduling latency. To minimize the latency due to running system services, we ran the system with only essential drivers and services. When a reply comes from the server, we simply forward it to the destination host without any modification.
2. If the protocol number in the IP packet is 6 (TCP), as before, we record the time at which the packet was received. We then inspect the TCP flag to determine its type.

    - If the received segment is a SYN (say with initial sequence number ISN_o) frame, then the source host is trying to establish a new TCP connection with the server. In this case, the SDMw system creates a new ISN (ISN_n), based on the underlying Linux kernel and records the mapping between <ISN_o, ISN_n> in the translational table. Except for the socket

information, SDMw modifies other TCP window parameters to reflect the underlying Linux behavior. The modified TCP segment is wrapped on to an IP packet, time-stamped and sent to the Server through CARD2.

- Whenever a reply (SYN-ACK for ISN_n) arrives from the server, based on the translational table, SDMw must translate the SYN-ACK for ISN_o.
- If a DATA segment with an implicit ACK is received from an inside host, then it must be a part of an on-going connection. The Sequence (SEQ) number for the DATA component must be changed by the SDMw using the translational table. This is to reflect the change in the ISN. SDMw adds (ISN_n-ISN_o) to the original SEQ number given by the host's TCP segment to reflect the new SEQ number.

  Since the SEQ number from the down-stream traffic (server to host) is unchanged by SDMw, the implicit ACK for the segment from the server is unchanged.
- If a DATA segment from the server is received addressed to an inside host, the SEQ numbers pertaining to the DATA are unchanged. However, any implicit ACK (for the up-stream traffic) from the server is changed to reflect the implicit ACK for the original TCP-DATA segment from the host using the translational table.
- If the received TCP-segment is a FIN segment from a host, similar to the DATA segment, the SEQ number for the FIN segment is changed to reflect the change in the ISN.

  Any FIN segment from the down-stream traffic is unchanged.

SDMw introduces the following latency for connection-oriented and connectionless IP services:

1. The protocol field of the incoming IP packet needs to be examined in order to perform the correct action.
2. For a connectionless IP or UDP services, only the IP-ID and flag fields needs to be changed. There is no need to maintain any reference table information.
3. For a connection-oriented TCP service, the latency depends on whether the incoming segment is a TCP-SYN or a DATA/ACK/FIN segment. For a new connection, SDMw must create a new ISN depending on the OS fingerprint in use, and create a mapping between the incoming ISN and the new ISN in the translational table. DATA/ACK/FIN segments are a part of an ongoing TCP connection. In this case, based on the socket information, SDMw searches the table finds the <ISN_o, ISN_n> mapping. Based on this mapping, the Sequence number field in the TCP segment must be updated. For DATA/ACK/FIN segments, SDMw introduces table-lookup latency along with Sequence number updating latency.

We collected statistics for 200 ICMP packets (100 from each host) and 200 TCP-DATA/ACK/FIN segments. We also collected the latency associated with TCP-SYN segments associated with the data connection. The performance graph is presented in Figure 4. As it can be seen from the performance graph, the SDMw system did not introduce considerable latency due to the header rewriting process. For TCP-DATA/ACK/FIN segments, the average latency is about 2.2 ms. For the associated TCP-SYN segments, the latency is 2.3 ms. For connectionless ICMP packets, the latency is 1.1 ms. The latency can be improved if we run the SDMw algorithm in an application-specific integrated circuit (ASIC) processor with highspeed content addressable memory.

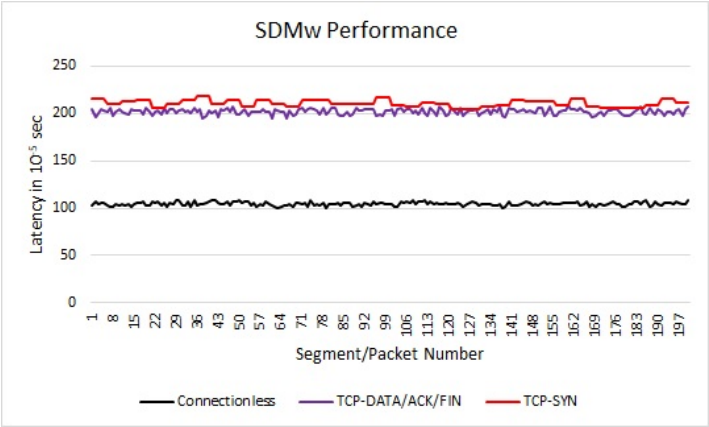

Figure 4. SDMw performance.

6. Conclusions

In this paper, we proposed a novel idea to misguide an attacker during the fingerprinting process. We call our proposed system SDMw–Secure Dynamic Middleware. Our proposed system has several advantages compared with similar models or systems that attempt to prevent fingerprinting by hackers:

1. Systems that try to defeat fingerprinting or hide the network particularly from NMAP expose themselves to fingerprinting. However, our proposed system misguides the attacker by showing different OS characteristics at different times.
2. The SDMw system does not introduce severe latency. Since the SDMw system is rewriting the header, NAT can be incorporated with SDMw. In addition, SOCKS5 can be integrated with SDMw as well.
3. The SDMw system may be used effectively as a Firewall. Since the SDMw system acts like a proxy, it can inspect incoming packets and decide whether to allow the packet into the external network or not. Unlike traditional firewalls, our proposed SDMw system can implement firewall rules based on either a single machine or a port or groups of hosts.

Our future work involves prototyping the SDMw system and evaluating its performance in a large enterprise network, implementing SDMw on a FPGA-chip to evaluate its core-performance, and optimizing the SDMw algorithm using intelligent cache algorithms.

Acknowledgments: The authors would like to thank the anonymous reviewers and the editorial team for their helpful and constructive comments that greatly contributed to improving the quality of our paper.

Author Contributions: The initial Solution architecture was conceived by Ms.Dalal Hanna as part of her honours thesis under the supervision of Ben Soh and Prakash Veeraraghavan. This work was further enhanced to include performance benchmarking, prototyping in 2017 by Dalal Hanna and Prakash Veeraraghavan.

Conflicts of Interest: The authors declare no conflicts of interest.

References

1. Cisco, Global IP Traffic Forecast and Methodology, 2006-2011, Available online: http://www.hbtf.org/files/cisco_IPforecast.pdf (accessed on 25 September 2017).
2. Davidoff, S.; Ham, J. *Network Forensics: Tracking Hackers through Cyberspace*; Prentice Hall Publication: Upper Saddle River, NJ, USA, 2012.
3. Gordon Foydor Lyon. Nmap: The Network Mapper-Free Security Scanner. Available online: https://nmap.org (accessed on 25 September 2017).
4. Kohno, T.; Broido, A.; Claffy, K. Remote physical device fingerprinting. *IEEE Trans. Dependable Secur. Comput.* **2005**, *2*, 93–108.

5. Shamsi, Z.; Nandwani, A.; Leonard, D.; Loguinov, D. Hershel: Single-Packet OS Fingerprinting. *IEEE/ACM Trans. Netw.* **2016**, *24*, 2196–2209.
6. Gordon Foydor Lyon. *The Art of Port Scanning*; Phrack Magazine, 1997; Volume 7. Available online: http://phrack.org/issues/51/11.html#article (accessed on 25 Septmcer 2017).
7. Gordon Foydor Lyon. *Nmap, Network Scanning*; Pub. Insecure. Com: Sunnyvale, CA, USA, 2008.
8. Dokas, P.; Ertoz, L.; Kumar, V.; Lazarevic, A.; Srivastava, J.; Tan, P. Data mining for network intrusion detection. In Proceedings of the NSF Workshop on Next Generation Data Mining, Baltimore, MD, USA, 1–3 November 2002; pp. 21–30.
9. Soniya, B.; Wisey, M. Detection of TCP SYN scanning using Packet counts and neural network. In Proceedings of the IEEE International Conference on Signal Image Technology and Internet Based Systems, SITIS, Bali, Indonesia, 30 November–3 December 2008; pp. 646–649.
10. Baig, H.U.; Kamran, F. Detection of Port and network scan using time independent feature set. In Proceedings of the IEEE Conference on Intelligence and Security Informatics, New Brunswick, NJ, USA, 23–24 May 2007; pp. 180–184.
11. Leckie, C.; Kotagiri, R. A probabilistic approach to detecting network scans. In Proceedings of the Eighth IEEE Network Operations and Management symposium (NOMS), Florence, Italy, 15–19 April 2002; pp. 359–372.
12. Fall, K.R.; Richard Stevens, W. *TCP/IP Illustrated, Volume 1: The Protocols*; Addison-Wesley Professional Computing: Boston, MA, USA, 2012.
13. Naga Raju, P. State-of-Art Intrusion Detection: Technologies, Challenges and Evaluation. Master Thesis, Linkoping University, Linköping, Sweden, 2005.