symmetry

Article

An Intelligent Improvement of Internet-Wide Scan Engine for Fast Discovery of Vulnerable IoT Devices

Hwankuk Kim, Taeun Kim and Daeil Jang *

Korea Internet & Security Agency, 9, Jinheung-gil, Naju-si, Jeollanam-do 58324, Korea; rinyfeel@kisa.or.kr (H.K.); tekim31@kisa.or.kr (T.K.)

* Correspondence: dale@kisa.or.kr; Tel.: +82-61-820-1274

Received: 31 March 2018; Accepted: 7 May 2018; Published: 10 May 2018

Abstract: Since 2016, Mirai and Persirai malware have infected hundreds of thousands of Internet of Things (IoT) devices and created a massive IoT botnet, which caused distributed denial of service (DDoS) attacks. IoT malware targets vulnerable IoT devices, which are vulnerable to security risks. Techniques are needed to prevent IoT devices from being exploited by attackers. However, unlike high-performance PCs, IoT devices are lightweight, low-power, and low-cost, having performance limitations regarding processing and memory, which makes it difficult to install security and anti-malware programs. Recently, several studies have been attempted to quickly search for vulnerable internet-connected devices to solve this real issue. Issues yet to be studied still exist regarding these types of internet-wide scan technologies, such as filtering by security devices and a shortage of collected operating system (OS) information. This paper proposes an intelligent internet-wide scan model that improves IP state scanning with advanced internet protocol (IP) randomization, reactive protocol (port) scanning, and OS fingerprinting scanning, applying k* algorithm in order to find vulnerable IoT devices. Additionally, we describe the experiment's results compared to the existing internet-wide scan technologies, such as ZMap and Shodan. As a result, the proposed model experimentally shows improved performance. Although we improved the ZMap, the throughput per minute (TPM) performance is similar to ZMap without degrading the IP scan throughput and the performance of generating a single IP address is about 118% better than ZMap. In the protocol scan performance experiments, it is about 129% better than the Censys based ZMap, and the performance of OS fingerprinting is better than ZMap, with about 50% accuracy.

Keywords: IoT; security; machine learning; vulnerability; intelligent security

1. Introduction

Gartner, Inc. forecasts that 8.4 billion connected things will be in use, worldwide, in 2017, up 31% from 2016, and will reach 20.4 billion by 2020. The total spending on endpoints and services will reach nearly $2 trillion in 2017 [1]. Meanwhile, it has become a reality that vulnerable IoT devices (CCTV, etc.) are frequently involuntarily involved in DDoS (distributed denial of service) attacks. In October 2016, the DNS service provider Dyn took down hundreds of websites—including Twitter, Netflix, and The New York Times—for several hours, due to IoT devices being infected with Mirai malware. As shown in Figure 1, Mirai malware primarily spreads by first infecting devices such as webcams, DVRs, and routers. It then deduces the administrative vulnerabilities of other IoT devices by means of brute force attack. Mirai mutations, such as Persia Lee, Ripper, and Bricker are generated daily [2,3].

The cause of IoT device infection by malware is mainly from security vulnerability management. This means IoT devices are used as cut-down OS, with no security functions, or are simply operated with a default ID and password. Cisco (2016) reported that more than 90% of network devices were running with known vulnerabilities, and there were 28 vulnerabilities per device on average [4].

In addition, according to a report by HP (2015), 80% of IoT devices have a default password that is vulnerable to privacy leakage, 70% of them are not encrypted during communication, and 60% of them have a weak web interface, and have not been updated for security [5].

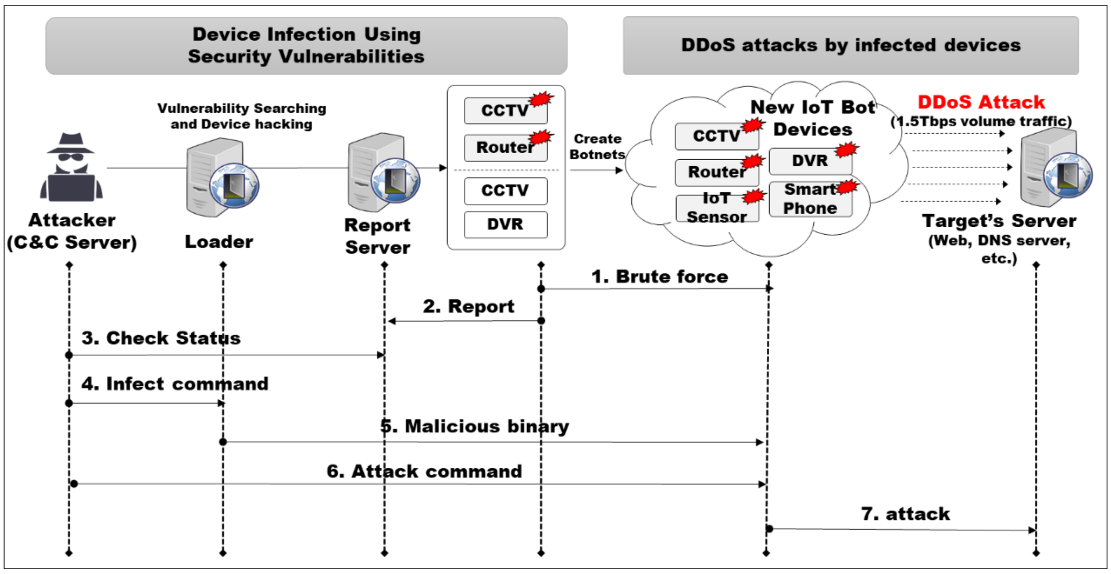

Figure 1. Operation of Mirai malware and DDoS (distributed denial of service) attacks by infected IoT botnet.

Today, there have been various "security by design" approaches for design-secure IoT devices and applications [6], and intelligent and secure monitoring of hierarchical topology smart home appliances and environments [7,8] by applying lightweight encryption, authentication, secure communication, and intrusion detection [9–12]. The most general approach is to install an antivirus program in an IoT device. However, since IoT devices are not high-performance devices, like PCs, or mobile devices, it is not easy to eliminate security vulnerabilities and manage periodic updates by installing an antivirus program or other malware detection technology. Additionally, low-end devices are incapable of performing heavier, conventional cryptographic algorithms, due to their constrained resources [13,14].

From the viewpoint of the security vulnerability management of IoT devices, there are various reasons as to why it is difficult to eliminate security vulnerabilities. First, since IoT devices have been operating for a long period of time by using open source code and old communication technologies, they are running with old vulnerabilities. Second, since IoT devices have hard-coded software or have no automatic update features, like PCs, when vulnerabilities in the embedded OS and firmware are detected, it is difficult to patch them quickly. Third, since IoT device manufacturers are small enterprises, if the company goes down, it is difficult to provide follow-up services, such as a security update, and security vulnerabilities are left unresolved [15].

Recently, there have been many studies on internet-wide vulnerability scanning, such as ZMap [16] and Shodan [17], where connected vulnerable IoT devices are scanned in real time, and stored in a database, and their search results are shared. In this study, we propose a model designed to collect IoT device information in order to detect vulnerabilities in IoT devices connected to the internet, and compare its performance with existing technology. The structure of this paper involves Section 2 explaining the concept and existing studies on passive vulnerability scanning technology. Section 3 presents the passive vulnerability scanning engine model, which has been improved from the open source ZMap, and explains the scanning algorithm and OS fingerprinting identification and classification algorithm. Section 4 explains the method of measuring its performance and the results. Section 5 presents a comparison with existing technologies, such as ZMap and Shodan, and explains future study.

2. Related Works

2.1. The Concept of Internet-Wide Vulnerability Scanning

IoT malware targets vulnerable IoT devices for the purposes of hacking and malware infection. To avoid such malware infection, the technique of scanning plays a key role in eliminating vulnerabilities of low-end IoT devices with constrained resources. Until now, vulnerability scanning techniques [18] have developed from network scanning in the late 1990s, which scans local network and system vulnerabilities, to internet-wide scanning, which detects vulnerable internet-connected devices regularly.

Figure 2 and Table 1 show the concept and features of traditional network scanning and internet-wide scanning [19]. The biggest difference of the two approaches is in the scanning method used to collect device information and scan vulnerabilities. In this paper, the viewpoint of distinguishing the network and internet-wide scanning is whether the technique uses a crafted packet to collect a device's information and find vulnerabilities. In other words, we defined that a scanner using a crafted packet is a network scan, since it is very intrusive, while using banner grabbing (or OS fingerprint) data is an internet-wide scan (non-intrusive internet-wide scan), since it is relatively less intrusive.

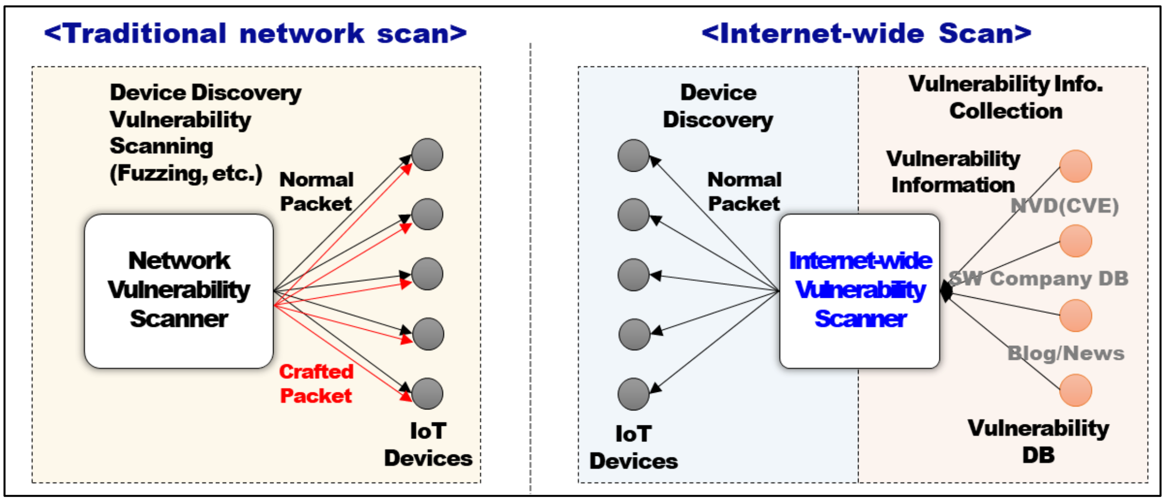
<Traditional network scan>
Device Discovery Vulnerability Scanning (Fuzzing, etc.)
Normal Packet
Network Vulnerability Scanner
Crafted Packet
IoT Devices
<Internet-wide Scan>
Device Discovery
Normal Packet
Internet-wide Vulnerability Scanner
IoT Devices
Vulnerability Info. Collection
Vulnerability Information
NVD(CVE)
SW Company DB
Blog/News
Vulnerability DB

Figure 2. Differences between traditional network scanning and internet-wide scanning techniques.

Table 1. Characteristic of network/internet-wide scan techniques.

Techniques	Traditional Network Scan	Internet-Wide Scan
Scan Range	Devices connected to local private network	Internet-wide devices
Characteristic	Very intrusive scan	Less intrusive (Non-intrusive) scan
Scan Method	Authorized User → Credential Scan (PW crack, fuzzing, and crafted packets)	Unauthorized User → Non Credential Scan (banners and normal messages)
Vulnerability Analysis	Known/Unknown class vulnerability analysis (network/service level)	Device information DB → Known class vulnerability detection
Analysis Method	Dynamic/static fuzzing analysis	Vulnerability DB
Related Technology	Nmap, Masscan, Nessus, etc.	Shodan (ShoVAT), ZMap (Censys)

The traditional network scan technique checks IPs for local networks to identify OS types, and collects the service type and version information through a port scan to detect vulnerabilities. To identify vulnerabilities, Nmap, Nessus, and other tools are used in network scanning. This technique attacks the device to find its vulnerabilities. For example, it uses crafted packets—well-known ID and password combinations—to check the vulnerability of the password, or directly attacks the device using an exploit code. However, since this technique collects internet-connected device information remotely, it could be filtered out by security equipment, such as a firewall. This is because the network scan techniques have the risk of shutting down the service or device to be inoperable by generating attacking patterns or traffic to the inspected device.

Therefore, many studies focus on internet-wide scanning techniques to collect a large volume of device data quickly from a remote location. Internet-wide scanning does not perform an attack to collect device information, but communicates regularity messages to collect necessary information. Furthermore, this technique covers all internet-connected devices, rather than only the inner network, and collects service access banners and communication traffic headers.

2.2. Preliminary Research Related to Internet-Wide Scanning

John Matherly [17,20] developed the Shodan search engine in order to search for information of internet-connected devices through a non-intrusive scan technique. Shodan collects the data of more than 500 million devices and systems on the internet and provides the collected data each month, but it is difficult to analyze the scan processing speed and method, since it is not disclosed as an open source, except for the list of application programming interfaces (APIs). Its main procedure is the scanning of information for HTTP, FTP, TELNET, and other open ports via handshaking. Device information is identified via keywords included in banners, and various protocols are supported to collect as much information as possible from a single device. Additionally, SSL encryption and version information are collected to discover heart bleed, poodle, and other vulnerabilities.

Shovat [21] is a passive vulnerability analysis tool developed by Petru Maior University in Romania with the Shodan engine. Shovat takes the output of traditional Shodan queries and performs an in-depth analysis of service-specific data, such as service banners. It embodies specially crafted algorithms that rely on novel in-memory data structures to automatically reconstruct common platform enumeration (CPE) names, and to proficiently extract vulnerabilities from the national vulnerability database (NVD) [22].

ZMap [16] is open source software, suggested by the University of Michigan, that can scan all internet-connected IPv4 address spaces, and applies random algorithm and sharing techniques to IP addresses for rapid searching. In addition, it is faster than Nmap, since it does not go through TCP/IP stacks. It can reportedly scan the 10 G IPv4 address space in 4 min and 29 s.

Censys [23] is based on the open source ZMap, and collects port information for 23 protocols. It provides banner information, protocol header information, other device information, and encrypted communication protocol vulnerabilities, just as Shodan does, but it does not have a technique for identifying OS information. The main procedure of Censys is to check whether the device IP is enabled, or the port is opened using a TCP SYN scan from a ZMap module. The ZGrab module, which is a plug application scanner, performs a handshake for banner grabbing if there is a response from a scanned device. It collects the data of the application of the port. Censys then extracts the relevant fields from the collected banner data, adds the metadata as a comment, and saves it in the database.

3. Proposed Model

This section describes the proposed model to improve the device collecting performance of ZMap and Shodan, which are some of the most effective internet-wide scanning techniques.

As shown in Figure 3, this model consists of three modules—IP alive scan module, reactive handshake scan module, and OS fingerprinting module. The IP alive scan module collects IoT device status information. The reactive protocol scan module collects network and service information, and the OS fingerprinting scan module collects OS information. To create a non-intrusive scan effect, the reactive handshake module does not store the TCP connection status data (no per-connection state) to find the protocol (port) data of devices. In the OS fingerprinting module, we applied the passive fingerprinting technique of using the TTL and window size values of the TCP/IP header.

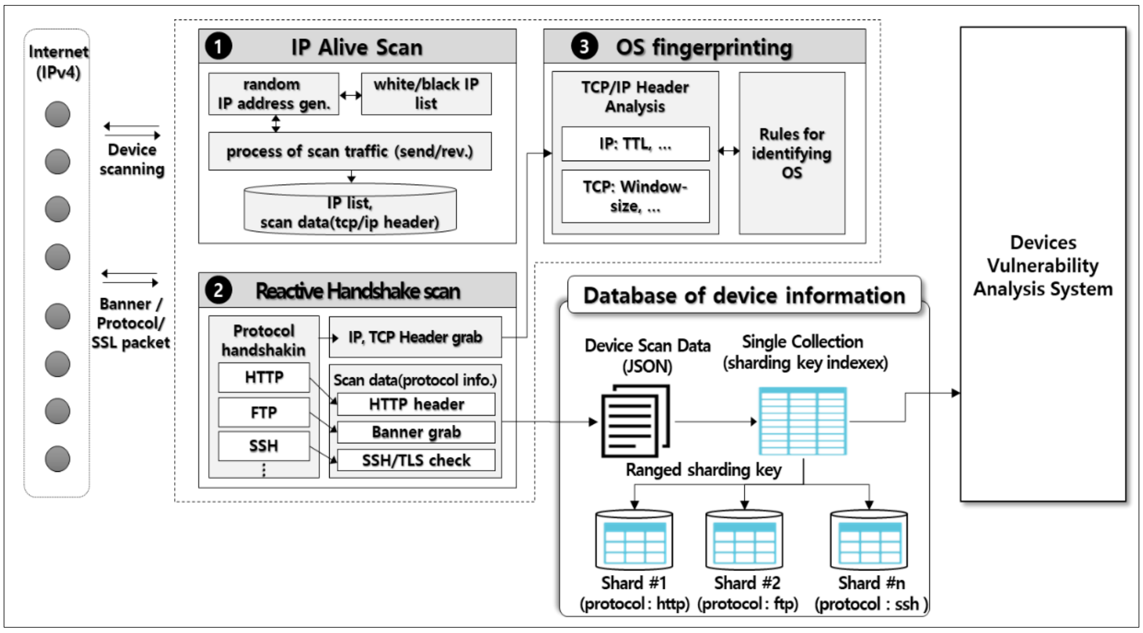
Internet (IPv4)
Device scanning
Banner / Protocol/ SSL packet
1 IP Alive Scan
random IP address gen.
white/black IP list
process of scan traffic (send/rev.)
IP list, scan data(tcp/ip header)
2 Reactive Handshake scan
Protocol handshakin
HTTP
FTP
SSH
IP, TCP Header grab
Scan data(protocol info.)
HTTP header
Banner grab
SSH/TLS check
3 OS fingerprinting
TCP/IP Header Analysis
IP: TTL, ...
TCP: Window-size, ...
Rules for identifying OS
Database of device information
Device Scan Data (JSON)
Single Collection (sharding key indexex)
Ranged sharding key
Shard #1 (protocol : http)
Shard #2 (protocol : ftp)
Shard #n (protocol : ssh)
Devices Vulnerability Analysis System

Figure 3. Proposed model of vulnerability scanning engine.

Our model basically improved the open source ZMap. Compared with the ZMap, it consists of two parts: a probe engine for fast scan to check IP activation, and the ZGrab engine to collect the application data (protocol or port) in a banner grabbing technique. The IP alive scan module enhanced the scheduler and IP address randomization of the ZMap's probe module. The reactive handshake scan module enhanced the ZGrab module, only collecting 23 examples of protocol banner information, theoretically extending the collection range to 63,568 port information. The ZMap engine does not provide the OS fingerprinting module—they only use banner grabbing with pattern matching—thus, we added TCP/IP stack fingerprinting techniques to identify the OS's type.

After the scanning finished, the collected data was stored in the DB for the device's information, and accumulated about 950 Gbytes per scan. Therefore, we used an Amazon cloud server to efficiently handle network traffic, instead of a single server to handle large amounts of device scan data. We also used Hadoop's HDFS and NoSQL (MogoDB) to multiplex the DB servers and distribute the data across multiple servers using "sharding" technology. In more detail, this method used the shard key to partition the collected data based on ranges of the key in order to store very large data sets. Each range of the shard key values is assigned to a specific chunk on the cluster. The collected data is stored in the JSON file format per IP address. The collected data are split over the shards using a defined range-based shard key, in order to store very large data sets. Here, the shard key is defined as protocol_id.

3.1. IP Alive Scan Module

The "IP alive scan module" performs functions to check whether the device is in active status by generating TCP SYN or ICMP packets. It consists of a random IP address generator and an IP scan scheduler. To improve the hit rate of response packets, the white list/black list is applied to the scheduler. An IP alive scan scheduler generates an address list to scan about 4.3 billion IPv4 addresses.

The technology of generating IP addresses is very important in internet-wide scan technology. When scan traffic is generated sequentially, the scan function is not available, due to the detection by security devices, such as firewalls, IDS, and IPS. Therefore, it is necessary to have a technology that randomly generates lists of IP addresses for scanning. Therefore, we used an algorithm that converts IP addresses to decimal numbers, and then circulates them to generate a randomized IP address list. Randomized IP addresses make them look random in the network domain.

As shown in Algorithm 1, This algorithm selects an initial decimal number from 2^{32} numbers, which is the size of the IPv4 address domain. A prime number between 2^{16} and 2^{32} is selected to generate an IP address that has the difference of B class or more. The prime number is subtracted from the initial number to generate a decimal number. The value 2^{32} is added to the initial number to generate a number between 1 and 2^{32} if the initial number is smaller than the prime number. The generated decimal number is converted to an IP address and used as the scan address. This algorithm can reduce duplication of IP addresses in B class level. Its purpose is to bypass scan traffic detection by a security system when the highest level bandwidth of the agency allocated of the IP is B class, and it is assumed that the security system is positioned upstream of the network.

Algorithm 1: The Randomization of IP Address Generation

```
function Random IP Generation();
begin
    a ← random.randrange(1, 4294967295);
    count ← 0;
    ipinit ← a;
    socket.inet_ntoa(struct.pack('!L', ipinit));
    while (count < 65535) do
        if a < 16583719 do
            a ← 4294967295 + a
        end if
        a ← a − 16583719;
        ipinit ← a;
        b ← str(socket.inet_ntoa(struct.pack('!L', ipinit)));
        count = count + 1;
    end while
end
```

As another technique to avoid filtering by security devices, we modified the scan scheduler to use WHOIS lookup information as a whitelist of IP address ranges for each domain assigned by ICANN (Internet Corporation for Assigned Names and Numbers). The scan scheduler generates IP lists based on the IP range of each domain, selects one of the IP lists, and scans the IP lists once in a specific time window. At this time, the representative IP of each domain is retrieved by searching the Whois lookup table, and the sub IP addresses of the corresponding domain are extracted. Then, the extracted IP list is shuffled using a randomized algorithm to prevent sequential scanning.

3.2. Reactive Handshake Protocol Scan Module

The "reactive protocol handshake scan" module collects the service (port) information running on its open ports through the IP list of alive states. This module scans information about major ports such as FTP, Telnet, SSH, and HTTP, that are used for communication in the device. Information collected from the major ports generates information by going through the process of extracting scan information, including the system connection banner, encrypted communication information, packet header, and HTTP header/body information.

Also, this technique is similar to the ZGrab module of the ZMap. The collection scope of this module identifies a total of 65,568 ports, of which 168 reserved ports and 65,400 unreserved dynamic ports were added to the fifteen basic protocol scan functions provided by ZGrab. Unreserved dynamic ports can be defined by the user or the input of extracted traffic data, such as PCAP. In addition, since most IoT devices connected to the wireless AP have a private IP, the m-search message of the

UPnP protocol (1900 port), and response information from NAS servers (9000 port) are used to collect the information of IoT devices connected to the wireless AP.

3.3. OS Fingerpriting Scan Module

The "OS fingerprinting scan module" is used to identify the OS and firmware information of internet-wide IoT devices. Our approach uses remote TCP/IP stack fingerprinting using TCP/IP headers and HTTP banner grabbing techniques. This module can be identified by comparing the OS matching rule with information from ten fields related to the TCP/IP packet as TCP Headers (don't_fragment, Window Size, MSS, Window Scale, Timestamps, NOP, sackOK) and IP Headers (IPID, Total Length, TTL). We applied the k^* classier algorithm to improve the accuracy of OS fingerprinting.

The k^* algorithm [24,25] is a simple instance-based classifier similar to the K-nearest neighbor (K-NN) classifier. A new data instance x is allocated to a class that occurs the most frequently in the k-nearest data point y_j, where j = 1,2 ... k [26]. Additionally, the entropy distance algorithm is used to search for the most similar instance in the data set. When the entropy distance is used as the metric, actual attributes and omissions can be processed along with other benefits [27]. The k^* algorithm is shown below.

$$k^*(y_i, x) = -lnP^*(y_i, x) \tag{1}$$

4. Experimental Results

4.1. Summary of Experiment Methods

In these experiments, two approaches were applied for testing the performance of the proposed model. First, we experimented with the scan throughput of the IP alive scan module in a test environment with no real traffic. Second, the rest of experiment was conducted by installing our model in Amazon cloud server. We collected 226 million pieces of real information of internet-connected devices from September to November 2017, and then measured performances which are the information collection rate and the OS identification rate. The experiment results were compared with various existing techniques, i.e., ZMap-based and Nmap-based Masscan (mass IP port scanner [28]) and Shodan.

4.2. IP Alive Scan Performance in the Test Environment

4.2.1. Throughput per Minute of IP Alive Scan

This technology focusing on public IPv4 addresses and collecting information about internet-connected devices requires high-speed traffic processing. The performance metric is TPM (throughput per minute) which generates 3.68 billion IP alive packets in the 1 Gigabit Ethernet environment, excluding private and reserved IPv4 addresses without system failure or errors.

$$\text{TPM} = \text{PPS}(\text{Generated Packets per Second}) * 60 \tag{2}$$

For the performance of the IP alive scan module, we used the BigTao tester, IP/ethernet testing equipment that can generate L2/L3 layer packets and measure the throughput, frame loss rate, and latency. The testing network environment is constructed by connecting the BigTao tester and DUT (device under test), as shown in Figure 4. We installed the proposed model (IP alive scan module), ZMap (version 2.1.1), and Masscan (1.0.4) in the DUT system. The DUT is configured to allow packet forwarding, and the network transmission and receipt interfaces of the DUT system are connected to the traffic receipt and transmission interfaces of the test system, respectively. These experiments also measure the performance of TPM according to the IEEE RFC 3511 procedure [29] in the environment that generates the background traffic. The BigTao tester is configured to receive the IP alive scan packets generated by the DUT, and the detailed procedure is as follows.

(1) BigTao tester generates 500 Mbps traffic to the DUT.
(2) DUT generates 3.67 billion scan packets (TCP SYN/ICMP) for each of the three technologies.
(3) BigTao tester checks the total number of frames collected during a period of 10 min.
(4) TPM is calculated based on the total number of frames.

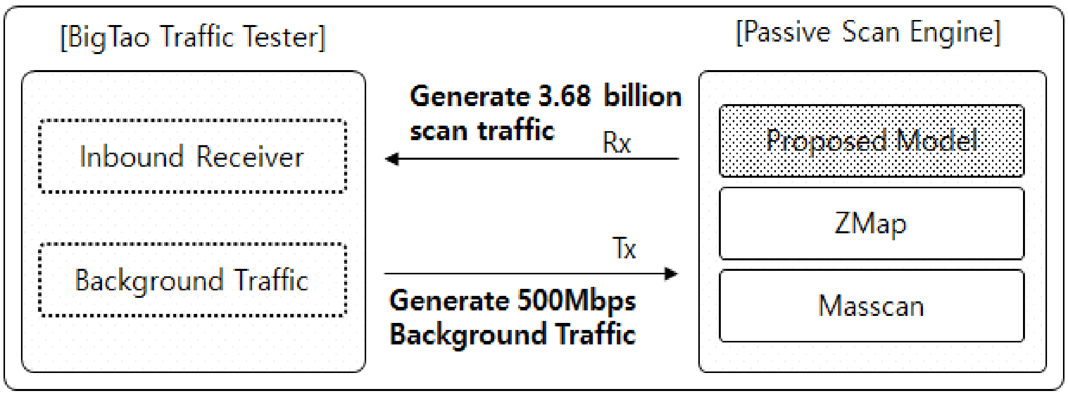

Figure 4. Experimental environment for measuring the throughput of scanning.

As shown in Figure 5 and Table 2, the experiment results indicate that the average scan speed was 85.94 million TPM in the no background traffic. This means that it took 50 min 9 s to scan the entire IPv4 range (4.3 billion IPs), and 44 min 8 s to scan the IPv4 range (3.68 billion IPs), excluding reserved IPs. With 500 Mbps background traffic, the scan speed showed almost no change. Therefore, in an environment where Rx and Tx are separated and when the hit-rate is 50%, scanning is possible without any change in speed.

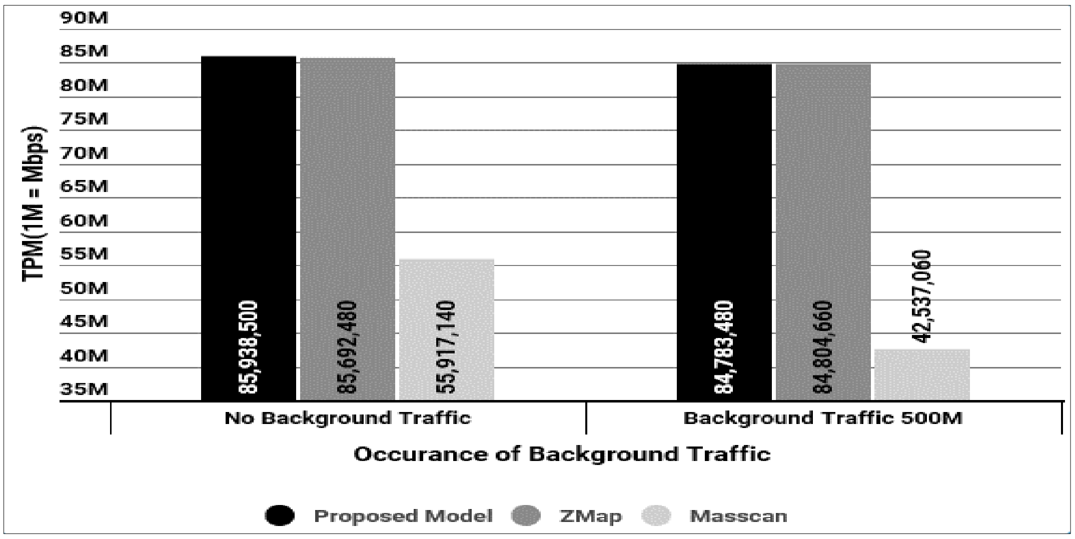

Figure 5. Comparison of the scan performance of existing tools.

Table 2. Comparison result with existing techniques (unit: TPMs).

Classification	Lab Environment (Back Traffic)		Real Internet Traffic
	0 M	500 M	
Our Model	85,938,500	84,783,480	72,750,000
ZMap	85,692,480	84,804,660	Unable to measure
Masscan	55,917,140	42,537,660	Unable to measure

This result shows that our model produces almost similar result as ZMap. This means that there was no performance degradation of IP alive scan, even after an enhancement (IP address randomization, etc.) of the ZMap. Our model and ZMap show better performance compared with Masscan (55.92 million TPM). Additionally, it was 72.75 billion TPM in the real internet environment. However, we did not perform a comparison experiment with existing techniques in this real internet environment.

4.2.2. Randomization of IP Address Generation

In this experiment, the performance metric is a randomization of IP address generation, in order to measure how many single IP addresses without duplicate IPs in the B/C IP classes were generated.

$$\text{Randomization of IP address generation } (\%) = \text{No.of Single IPs/B class} \tag{3}$$

As shown in Table 3 and Figure 6, this result indicates that 33,597 single IP addresses were generated (67.8%) for B class IPv4 addresses, and 65,530 single IP addresses were generated (99.99%) for C class IPv4 addresses. This means, in theory, that it took 14 min and 44 s to generate 3.67 billion IPv4 addresses and, therefore, 4,858,560 IP addresses could be generated per second. In addition, the randomization of the proposed model was 68%, which is an improvement of 118% compared to ZMap (58%) and 126% compared to Masscan (54%).

Table 3. Compared with existing techniques related to single IP generation rate.

Classification	Single IP	2-Duplicate IP	3-Duplicate IP	4-Duplicate IP	5 or More-Duplicate IP
Our Model	67.8%	32.2%	0.0%	0.0%	0.0%
ZMap	57.7%	29.5%	9.7%	2.5%	0.6%
Masscan	53.7%	27.5%	11.3%	4.7%	2.9%

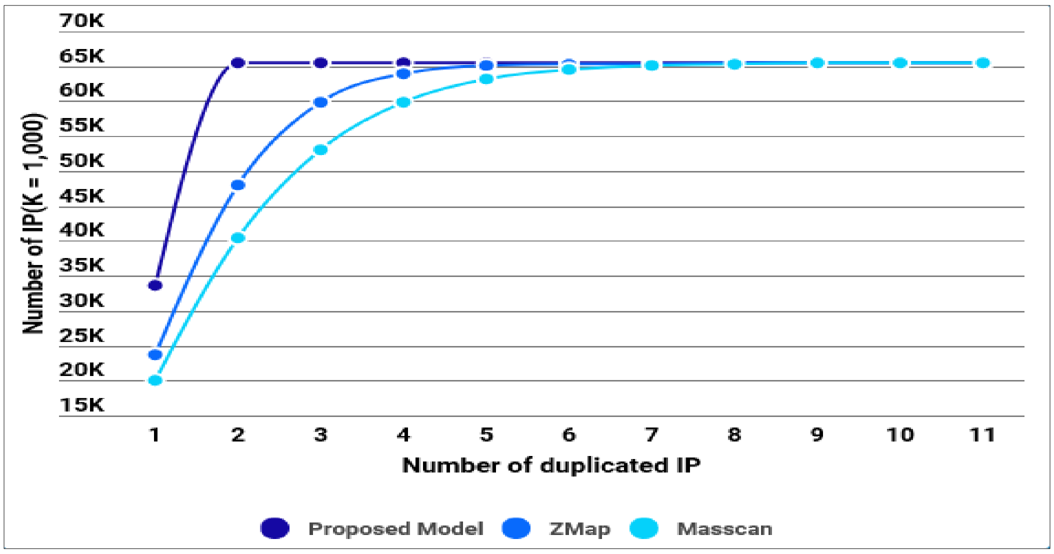

Figure 6. Result of single IP address generation.

4.3. Average Protocol (Port) Information Collection Rate (%)

The reactive scan module identifies the protocol service information from the opened ports of the activated IPs collected by the IP alive scan. In this experiment, the performance metric is the average protocol (port) information collection rate (PICR) and compared it with ZMap and Censys, which is shown in Figure 7. In this experiment, we used 195.68 million pieces of IP data (excluding the duplicates) collected through the IP alive scan between September and November 2017. On the other hand, the performance measurement of existing techniques uses the total sum of searched IPs per each port using the API supported by the Shodan and Censys services, respectively.

$$\text{Average PICR } (\%) = \frac{\sum_{i=1}^{n} \frac{K_i}{S_i|C_i} * 100}{n} \tag{4}$$

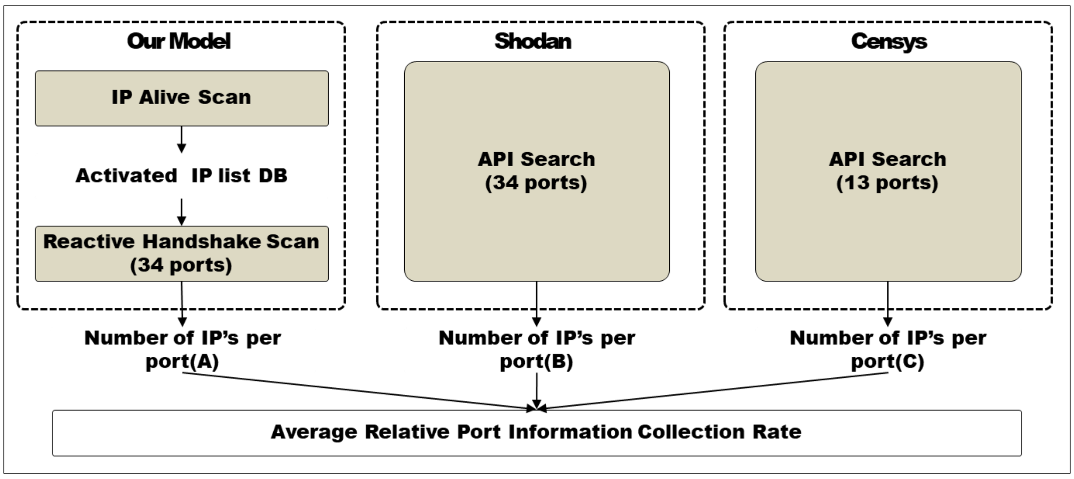
Our Model
IP Alive Scan
Activated IP list DB
Reactive Handshake Scan (34 ports)
Shodan
API Search (34 ports)
Censys
API Search (13 ports)
Number of IP's per port(A)
Number of IP's per port(B)
Number of IP's per port(C)
Average Relative Port Information Collection Rate

Figure 7. Measurement method of port information collection rate with scanning techniques.

Equation (4) shows the calculation of the average PICR. In this equation, K_i = the number of collected IPs in each port of the proposed model, S_i = the number of IPs collected per each port searched with Shodan API, C_i = the number of IPs collected per each port searched with Censys API, and i = the reserved port count (1, . . . , n).

To compare the three technologies under the same condition, we used a total of 34 ports, which is the maximum number allowed by the search API of Shodan. Although the ZMap-based Censys technology can analyze up to 23 ports, we could search the data of only 15 ports under the same port condition provided by the search API of Shodan.

As shown in Table 4, The test results showed that the proposed model collected a total of 223,559,189 IPs (including duplicates) from 34 ports, while Shodan collected 212,428,355 IPs from 34 ports, and Censys collected 173,496,395 IPs from 13 ports. The average PICR of the proposed module was 105.24% and 128.86% compared to Shodan and Censys, respectively.

Table 4. Comparison of average PICR with existing techniques. The symbol "-" means that the service protocol information is not provided by Censys API.

Service Protocol	**Number of Collected IP**			**Average PICR**	
	Our Model (A_i)	**Shodan (S_i)**	**Censys (C_i)**	**A_i/S_i**	**A_i/C_i**
Echo	68,565	243,449	Not Supported	28.16%	-
Systat	3906	3322	Not Supported	117.58%	-
Daytime	84,151	45,628	Not Supported	184.43%	-
Netstat	3283	2677	Not Supported	122.64%	-
Quote of the day	33,683	28,729	Not Supported	117.24%	-
FTP	16,071,122	6,011,068	10,478,740	267.36%	153.37%
SSH	20,017,760	13,845,201	9,783,239	144.58%	204.61%
Telnet	8,789,068	5,759,225	3,910,163	152.61%	224.77%
SMTP	13,473,119	6,034,975	7,363,710	223.25%	182.97%
Finger	19,654	18,464	Not Supported	106.44%	-
HTTP	62,062,880	72,580,341	56,194,847	85.51%	110.44%
HTTP (81)	1,528,978	2,863,257	Not Supported	53.40%	-
HTTP (82)	509,387	967,095	Not Supported	52.67%	-

Table 4. *Cont.*

Service Protocol	Number of Collected IP			Average PICR	
	Our Model (A_i)	**Shodan (S_i)**	**Censys (C_i)**	A_i/S_i	A_i/C_i
HTTP (83)	270,148	388,326	Not Supported	69.57%	-
HTTP (84)	131,533	179,313	Not Supported	73.35%	-
Siemens S7	2,846,517	2655	4986	107,213.45%	57,090.19%
POP3	8,578,700	4,639,066	5,224,945	184.92%	164.19%
IMAP	8,763,105	4,143,730	4,643,772	211.48%	188.71%
HTTPS	47,471,574	54,942,697	49,711,654	86.40%	95.49%
Modbus	2,817,111	13,903	26,128	20,262.61%	10,781.96%
IMAP + SSL	7,781,633	3,674,578	4,399,639	211.77%	211.77%
POP3 + SSL	7,338,645	3,585,813	4,402,802	204.66%	204.66%
SIP	112	16,749,194	Not Supported	0.00%	-
Oracle HTTP	6018	28,527	Not Supported	21.10%	-
Zimbra HTTP	10,554	48,255	Not Supported	21.87%	-
HTTP (7657)	6259	22,284	Not Supported	28.09%	-
HTTP (8080)	6,738,602	11,328,440	17,334,893	59.48%	38.87%
Riak Web	101,923	161,546	Not Supported	63.09%	-
HTTP (8181)	605,813	849,940	Not Supported	71.28%	-
HTTPS (8443)	1,204,599	2,641,811	Not Supported	45.60%	-
MS-HTTPAPI	130,974	244,955	Not Supported	53.47%	-
DNP3	3,274,665	335,747	357	975.34%	917,273.11%
MongoDB	8693	31,077	Not Supported	27.97%	-
BACnet	2,806,455	13,067	16,520	21,477.42%	16,988.23%
Summary	223,559,189	212,428,355	173,496,395	105.24%	128.86%

4.4. Accuracy of OS Fingerprinting Identificatiion

In this experiment, the performance metric is the accuracy of OS fingerprinting identification, whether the OS was identified correctly. As explained in Section 4.1, we used real data (TCP/IP packets and http banner grabbing) collected from the real internet environment for performance measurement.

As shown in Figure 8, the datasets for training consist of $\{X_i, Y_i\}$ pairs, as follow in Equation (5). Here, Let N be the number of training input data, d be the number of features, and L be the label of OS information.

$$\text{Training Datasets} = \{X_i, Y_i\},\ i = 1, \ldots, N,\ X_i = \{f_1, f_2, \ldots, f_d\},\ Y_i \in \{L_1, L_2, \ldots, L_m\} \tag{5}$$

(1) Each X_i is a d dimensional feature vector extracted from TCP/IP headers, such as Window Size, MSS, Timestamps, NOP, sackOK, IPID, Total Length, TTL, etc.
(2) Each Y_i is the corresponding label information which uses OS information extracted from HTTP banner grabbing on the same IP.
(3) N is the total number of training input data, m is the number of OS types.

For the training datasets, we trained input data consisted of 21,000 TCP/IP raw packets and HTTP information (3000 samples each for seven OSs). In the test stage, the actual class (correct answer) to measure the accuracy of the OS identification used 7000 pieces of banner information (1000 samples each for seven OSs) for verification. We utilized a set of algorithms in the WEKA tool and trained 4 classifiers: k^* classifier, BF Tree, Random Tree, C4.5 for OS fingerprinting.

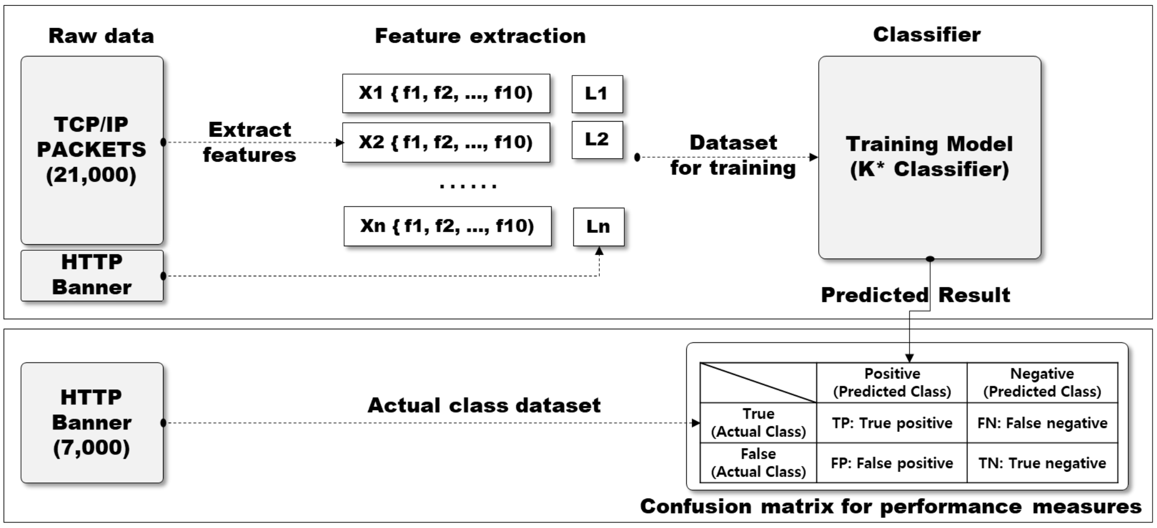

Figure 8. Experiment procedure of performance OS fingerprinting identification rate.

Figure 9 shows the OS identification accuracy results of the four classifiers. The accuracy (here means TPR) of the classifiers is in the order of *k**, c4.5, BFTree, and Random Tree, ranging from 0.477 to 0.491. Table 5 shows accuracy of each OS. Based on the *k** classifier, the average accuracy was 0.491, and the accuracy by OS was in order of FreeBSD (0.961), Gento (0.652), and Ubuntu (0.532), and Fedora had the lowest accuracy.

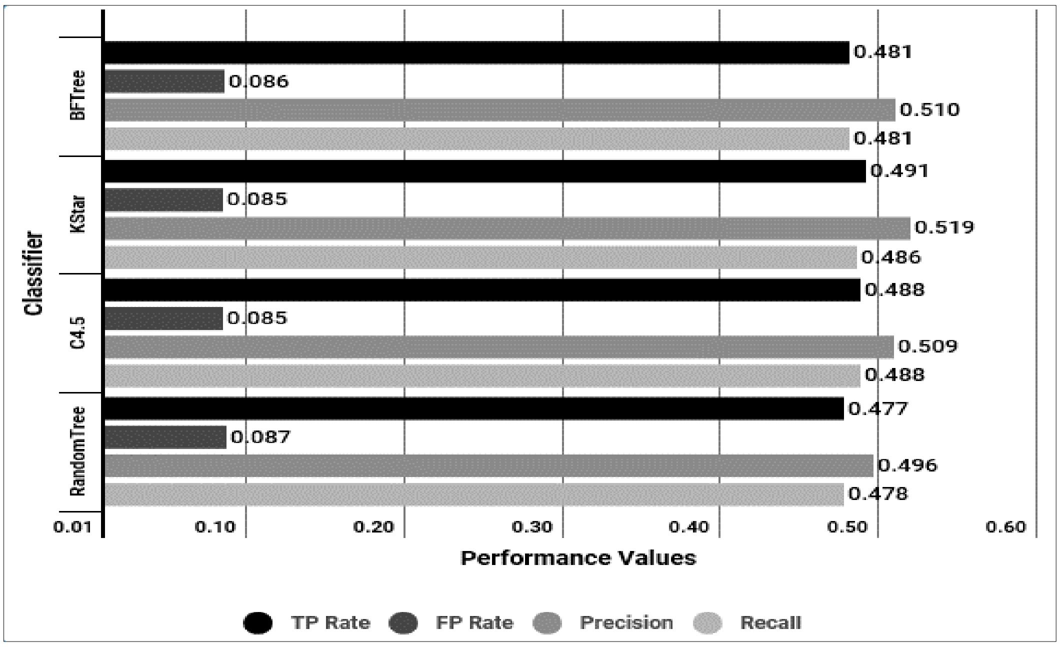

Figure 9. Comparison result of 4 classifiers (*k**, C4.5, BFTree, Random Tree).

Table 5. Results of OS identification test (TPR: True Positive Rate, FPR: False Positive Rate).

Classifier	BFTree		*k**		C4.5		RandomTree	
	TPR	**FPR**	**TPR**	**FPR**	**TPR**	**FPR**	**TPR**	**FPR**
Debian	0.216	0.1	0.245	0.091	0.248	0.11	0.226	0.118
Fedora	0.165	0.072	0.213	0.06	0.174	0.063	0.248	0.08
FreeBSD	0.965	0.006	0.961	0.009	0.965	0.006	0.961	0.006
Windows	0.49	0.015	0.452	0.017	0.516	0.02	0.513	0.027
Ubuntu	0.516	0.136	0.532	0.136	0.487	0.126	0.419	0.133
Gentoo	0.661	0.217	0.652	0.198	0.574	0.186	0.552	0.167
Redhat	0.356	0.06	0.385	0.082	0.45	0.087	0.417	0.079
Avg.	0.481	0.086	0.491	0.085	0.488	0.085	0.477	0.087

Figure 10 shows the results of comparing with TPR, FPR, precision, RoC, etc., in order to evaluate the accuracy of OS identification of the K star classifier. As a result, the precision was 0.519, TPR was 0.491, and FPR was 0.085. The RoC value that measures the utility of the model is 0.803, which is a good performance.

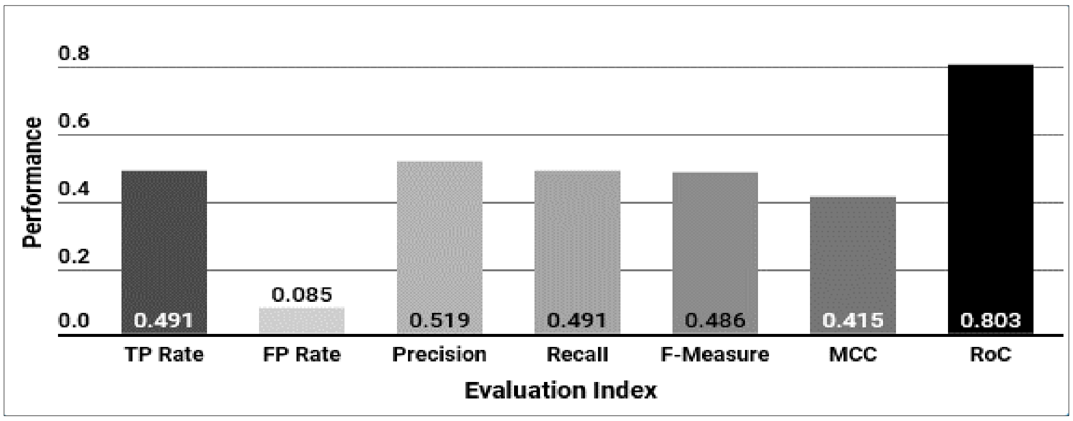

Figure 10. Confusion matrix for classification model performance.

According to previous research results [30,31], the average accuracy of exact matching without using machine learning is 15.87~31.25%, and the average accuracy of approximate matching is 2.87~3.1%. On the other hand, when applying machine learning, the accuracy was improved to almost 50~60%. This results are better than the pattern matching results, but they are different from previous studies using machine learning. This is because it is difficult to make a relative comparison and analysis, because the performance results depend on the experimental method, the characteristics of the learning dataset (number of collected training data, the collection method, bias of the training data, range of OS identification, etc.).

5. Conclusions

When vulnerable IoT devices are infected to become zombies, this can cause DDoS attacks. Therefore, it is important to scan vulnerable internet-connected IoT devices and take appropriate measures. There are many approaches to this, but this study proposed an internet-wide scan engine that can detect vulnerable IoT devices.

In this paper, we presented an improved internet-wide scanning model with various approaches, including the advanced algorithm of IP address generation, the extension of the range of protocols collection, and more accurate OS fingerprinting modules with machine learning. We conducted various experiments to measure the effectiveness of the proposed model and summarize the results, as shown in Table 6.

Table 6. Summary of comparison with our model and existing internet-wide scanning techniques. The term "unable to measure" could not be tested because it did not open the source.

	Features	**Our Model**	**ZMap/Censys**	**Masscan**	**Shodan (NMap Based)**
IP Alive Scan	Alive Scan Technique	TCP SYN ICMP	TCP SYN ICMP	TCP SYN ICMP	TCP SYN ICMP
	Throughput of Scanning	85,940,000 TPM	85,690,000 TPM	55,920,000 TPM	Unable to measure
	Single IP Address Generation Algorithm (Randomization)	Prime number-based Algorithm B class: 67.8% C class: 99.9%	Permutation Algorithm B class: 57.7% C class: 99.6%	Permutation Algorithm B class: 53.7% C class: 83.1%	Unable to measure
	IPv4 Address Generation Speed	14 min 44 s	18 min 3 s	19 min 9 s	Unable to measure
Protocol (port) Scan	Scan Technique	3 types (Banner GrabProtocol ErrorSeed File)	1 type (Banner Grab)	1 type (Banner Grab)	1 type (Banner Grab)
	Range of port collection	65,535 (168 protocols)	65,535 (23 protocols)	65,535 (17 protocols)	65,535 (168 protocols)
	Average PICR	223.57 million (34 ports) (105% > Shodan, 129% > Censys)	173.5 million (15 ports)	Unable to measure	212.42 million (34 ports)
	Scanning Time (Based on 130 million data sets each time)	Avg. 90 min (45 min for a single HTTP port)	Avg. 100 min	Avg. 102 min	Unable to measure
OS Fingerprinting Scan	Fingerprinting Technique	TCP·IP Stack Fingerprinting/Banner Grab	Banner Grab	Unable to measure	Banner Grab
	The number of Identifiable OS/Firmware	138 types (77 OSes, 61 Firmware)	PC OS	Unable to measure	Unable to measure
	Accuracy	51% (k* algorithm)	Pattern Matching	Unable to measure	Pattern Matching

As a result, this work is a contribution towards improving the capabilities of the current internet-wide scanning, by comparing with three performance metrics. First, the performance of IP alive scan was similar to ZMap in scanning throughput (TPM). However, the performance of generating a single IP address to prevent being filtered by security devices was about 118% (based on B Class) better than ZMap. Second, the protocol scanning collection performance showed that Censys based on ZMap can collect the information of 173.5 million IPs (based on 15 protocols), while the proposed model can collect the information of 223.57 million IPs, which was about 129% better than the Censys based ZMap. Third, the OS identification performance of ZMap is about 10~20% using exact matching technique, but the proposed model showed better performance than ZMap with about 50% identification accuracy. We carefully analyzed that these results experimentally showed that it seems effective, with results with improved performance compared to existing techniques.

On the other hand, this paper is for the improvement of internet-wide scanning technology, but there is a limitation in that the accuracy of device information identification is not high. In the future, additional research will be conducted to improve the accuracy of OS identification by applying deep learning for collecting device information at high speed.

Author Contributions: H.K. designed the system and wrote the paper. T.K. reviewed the paper and investigated related works. D.J. implemented the system and performed the experiments.

Acknowledgments: This work was supported by the Institute for Information & Communications Technology Promotion (IITP) with a grant funded by the Korean government (MSIT) (No. 2016-0-00193, IoT Security Vulnerabilities Search, Sharing and Testing Technology Development).

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

References

1. Gartner Newsroom. Available online: https://www.gartner.com/newsroom/id/3598917 (accessed on 7 February 2017).
2. KISA. 2016 Trend of Mirai Malware. Available online: https://www.krcert.or.kr/data/reportList.do (accessed on 7 February 2016).
3. Kolias, C.; Kambourakis, G.; Stavrou, A.; Voas, J. DDoS in the IoT: Mirai and Other Botnets. *Computer* **2017**, *50*, 40–41. [CrossRef]
4. Cisco Systems. *Midyear Security Report. Percentage of Devices Running Known Vulnerabilities by Age*; Cisco Systems: San Jose, CA, USA, 2016; p. 30.
5. HP Inc. HP Study Reveals Smartwatches Vulnerable to Attack. Available online: http://www8.hp.com/us/en/hp-news/press-release.html?id=2037386#.WmLdU6hl8dU (accessed on 22 July 2015).
6. IBM. Anatomy of an IoT Malware Attack. Available online: https://www.ibm.com/developerworks/library/iot-anatomy-iot-malware-attack/ (accessed on 31 October 2017).
7. Chang, S.H.; William, T. Design of an authentication and key management system for a smart meter gateway in AMI. In Proceedings of the 2017 IEEE 6th Global Conference on Consumer Electronics (GCCE), Nagoya, Japan, 24–27 October 2017.
8. Zhang, Y.; Xiang, Y.; Huang, X.; Chen, X.; Alelaiwi, A. A matrix-based cross-layer key establishment protocol for smart homes. In *Information Sciences*; Elsevier: New York, NY, USA, 2018; Volume 429, pp. 390–405.
9. Suryani, V.; Sulistyo, S.; Widyawan, W. Internet of Things (IoT) Framework for Granting Trust among Objects. *J. Inf. Process. Syst.* **2017**, *13*, 1613–1627.
10. Kim, M.; Lim, N.Y.; Park, J.H. A Security Generic Service Interface of Internet of Things (IoT) Platforms. *Symmetry* **2017**, *9*, 171. [CrossRef]
11. Kang, W.M.; Moon, S.Y.; Park, J.H. An enhanced security framework for home. In *Human-Centric Computing & Information Sciences*; Springer: Berlin, Germany, 2017; Volume 7, pp. 1–12.
12. Kim, M.S.; Lim, K.S.; Song, J.S.; Jun, M.S. An Efficient Secure Scheme Based on Hierarchical Topology in the Smart Home Environment. *Symmetry* **2017**, *9*, 143. [CrossRef]
13. Maity, S.; Park, J.H. Powering IoT Devices: A Novel Design and Analysis Technique. *J. Converg.* **2016**, *7*, 1–18.
14. Xiruo, L.; Meiyuan, Z.; Sugang, L.; Zhang, F.; Trappe, W. A Security Framework for the Internet of Things in the Future Internet Architecture. *Future Internet* **2017**, *9*, 27.
15. Kim, H.K.; Kim, T.E.; Ko, E.H. Management platform of threats information in IoT environment. In *Journal of Ambient Intelligence and Humanized Computing*; Springer: Berlin, Germany, 2017; pp. 1–10.
16. Durumeric, Z.; Bailey, M.; Halderman, J.H. An Internet-Wide View of Internet-Wide Scanning. In Proceedings of the 23rd USENIX Security Symposium, San Diego, CA, USA, 20–22 August 2014; pp. 65–78.
17. Shodan Project. Available online: https://www.Shodan.io/ (accessed on 10 December 2017).
18. Trapickin, R. Who is scanning the internet? In Proceedings of the Seminars Future Internet and Innovative Internet Technologies and Mobile Communications, Muchnich, Germany, September 2015; pp. 81–88.
19. Myers, D.; Foo, E.; Radke, K. Internet-wide scanning taxonomy and framework. In Proceedings of the Australasian Information Security Conference, Sydney, Australia, 21–30 January 2015; Australian Computer Society, Inc.: Sydney, Australia, 2015; pp. 61–65.
20. Matherly, J. Complete Guide to Shodan Collect, Analyze, Visualize, and Make Internet Intelligence Work for You. Available online: https://leanpub.com/shodan (accessed on 10 December 2017).
21. Genge, B.; Enachscu, C. ShoVAT: Shodan-based vulnerability assessment tool for Internet-facing services. *Secur. Commun. Netw.* **2015**, *9*, 2696–2714. [CrossRef]
22. National Vulnerability Database. NIST. Available online: https://nvd.nist.gov/ (accessed on 10 December 2017).
23. Durumeric, Z.; Adrian, D.; Mirian, A.; Bailey, M.; Halderman, J.A. A Search Engine Backed by Internet-Wide Scanning. In Proceedings of the 22nd ACM Conference on Computer and Communications Security, Denver, CO, USA, 12–16 October 2015; ACM: New York, NY, USA, 2015; pp. 542–553.
24. Hart, P. The condensed nearest neighbor rule. *IEEE Trans. Inf. Theory* **1968**, *14*, 515–516. [CrossRef]

25. Douglas, P.K.; Harris, S.; Yuille, A.; Cohen, M.S. Performance comparison of machine learning algorithms and number of independent components used in fMRI decoding of belief vs. disbelief. *Neuroimage* **2011**, *565*, 544–553. [CrossRef] [PubMed]
26. Aljazzar, H.; Leue, S. K*: A Heuristic Search Algorithm for Finding the k Shortest Paths. *Artif. Intell.* **2011**, *175*, 2129–2154. [CrossRef]
27. Clary, J.G.; Leonard, E.T. K*: An Instance-based Learner Using an Entropic Distance Measure. In Proceedings of the 12th International Conference on Machine Learning, New York, NY, USA, 16–21 July; Elsevier: New York, NY, USA, 1995; Volume 175, pp. 2129–2154.
28. Masscan Project. Available online: https://github.com/robertdavidgraham/masscan (accessed on 10 December 2017).
29. Hickman, B.; Newman, D.; Tadjudin, S.; Martin, T. IETF RFC 3511: Benchmarking Methodology for Firewall Performance. IETF, 2003. Available online: https://tools.ietf.org/html/rfc3511 (accessed on 5 January 2018).
30. Lee, H.S.; Kim, M.S. Research on OS fingerprinting method for real-time traffic analysis system. *J. Korea Inst. Commun. Inf. Sci.* **2011**, *36*, 443–450. [CrossRef]
31. Anderson, B.; Mcgrew, D. OS fingerprinting: New techniques and a study of information gain and obfuscation. *arXiv* **2017**.