PAPER

RPAH: A Moving Target Network Defense Mechanism Naturally Resists Reconnaissances and Attacks

Yue-Bin LUO^{†a)}, Bao-Sheng WANG[†], Xiao-Feng WANG[†], Bo-Feng ZHANG[†], *Nonmembers,*
and Wei HU[†], *Student Member*

SUMMARY Network servers and applications commonly use static IP addresses and communication ports, making themselves easy targets for network reconnaissances and attacks. Moving target defense (MTD) is an innovatory and promising proactive defense technique. In this paper, we develop a novel MTD mechanism, called Random Port and Address Hopping (RPAH). The goal of RPAH is to hide network servers and applications and resist network reconnaissances and attacks by constantly changing their IP addresses and ports. In order to enhance the unpredictability, RPAH integrates source identity, service identity and temporal parameter in the hopping to provide three hopping frequencies, i.e., source hopping, service hopping and temporal hopping. RPAH provides high unpredictability and the maximum hopping diversities by introducing port and address demultiplexing mechanism, and provides a convenient attack detection mechanism with which the messages from attackers using invalid or inactive addresses/ports will be conveniently detected and denied. Our experiments and evaluation on campus network and PlanetLab show that RPAH is effective in resisting various network reconnaissance and attack models such as network scanning and worm propagation, while introducing an acceptable operation overhead.

key words: port and address hopping, moving target defense, network security, reconnaissance

1. Introduction

Legacy networks use static IP addresses and well-known communication ports to provide standard network services. The benefit is that an end host can easily join the network and communicate with any other host connected to the network by exchanging packets. However, attackers can also take advantage of opportunities to perform network reconnaissances and launch attacks. Today, thousands of hosts in the Internet are becoming attack targets and suffering more and more network attacks and threats.

Moving target defense (MTD) [1] was proposed as an innovative proactive defense technique, which defends against attackers through continuously shifting systems' attack surfaces [2], [3] and increasing attackers' efforts and costs in exploiting target systems. As we all known, to design a system without any vulnerabilities is an unattainable goal. With the help of MTD techniques, system security will not strictly rely on totally eliminating system's vulnerabilities. By deploying MTD schemes such as instruction set randomization (ISR) [4], address space ran-

domization (ASR) [5], data space randomization (DSR) [6] and configuration randomization [7], existing system vulnerabilities will be well protected under the thought of diversity defense [8]. Although a few vulnerabilities exist, the system can maintain a relatively secure state. Even if an attacker holds an exploitable vulnerability, it is also difficult for him to successfully launch an attack to the varying system environment. A prominent advantage of MTD over traditional security techniques is that MTD is a promising active defense relying on proactive and constant variations other than rules or experiences of the past world. Therefore, MTD is of great value to change the current imbalance between network attacks and defenses.

In current networks, a standard network server host commonly provides a service by using a static IP address and opening a well-known port (e.g. 80 for web service) that will be kept open during the whole service lifetime. In this static service model, any client wants to connect the service can establish a connection using the static IP and port. However, the static service model also makes services under a long-term exposure to potential attackers. Attackers can perform network scanning, target the system, get detailed information, exploit potential vulnerabilities and launch DoS/DDoS [9] or some other advanced attacks to compromise the server system. Unfortunately, existing rule-based security systems such as firewall, IDS/IPS (Intrusion Detection/Prevention Systems) [10], [11] and antivirus softwares are insufficient to insure security because they are passive in nature, and they are incomplete and can only detect attacks known already.

Port and address hopping is a typical network MTD mechanism inspired by the technique of frequency hopping in wireless communications, which spreads the data stream of a communication session across multiple data connections named channels. The central idea of port and address hopping is to hide the real service identity by constantly changing communication ports and IP addresses so that to create high unpredictable network environments, where attackers need to expand much more efforts in their attack reconnaissances and preparations. An additional benefit of this tactic is that it increases the likelihood of detecting attackers. Since attackers cannot accurately locate the target service, they are forced to re-scan the target network frequently in different time. These recurring probes significantly slowing down the attack progress, while improving their detectability. Even if an attacker is capable of

Manuscript received July 13, 2016.

Manuscript revised October 12, 2016.

Manuscript publicized December 6, 2016.

[†]The authors are with the College of Computer, National University of Defense Technology, 410073, Changsha, China.

a) E-mail: luoyuebin@nudt.edu.cn

DOI: 10.1587/transinf.2016EDP7304

monitoring the network traffic and obtaining communication information at a given time, the information will become meaningless at the next time interval after a hopping event happens.

This paper describes a novel moving target network defense mechanism, called Random Port and Address Hopping (RPAH), which changes ports and IP addresses frequently and randomly to resist network reconnaissances and attacks. The basic ideal of RPAH is a time-synchronization based port and address hopping, but it enables servers to randomly vary their communication ports and IP addresses based on source identity, service identity as well as time. Besides, we design an address and port demultiplexing mechanism and a rigorous access control mechanism to ensure the hopping unpredictability using the entire address and port space, which introduces enormous hopping diversities to increase its defense ability. We evaluate the effectiveness and overhead of RPAH through actual implementation on our campus network as well as the *PlanetLab* platform, which shows that RPAH can reduce the accuracy of information gathering effectively, and it is very effective in thwarting various network reconnaissance and attack models such as network scanning and worm propagation, while introducing an acceptable operation overhead. This paper offers the following contributions:

- RPAH integrates source identity, service identity and time as the main hopping parameters, providing three hopping frequencies and offering significant effectiveness over previous port or/and address hopping techniques.
- RPAH provides a practical technique for host and service hiding by performing port and address hopping without interrupting active sessions during the hopping operations.
- RPAH introduces a port and address demultiplexing mechanism, providing high unpredictability and the maximum hopping diversities by using the whole port and address space.
- RPAH provides a convenient attack detection and access control mechanism, with which attackers that do not know the active ports/addresses cannot reach the service and connection attempts from attackers using invalid or inactive ports/addresses will be detected and denied.

The rest of this paper is organized as follows. Section 2 reviews the related work. In Sect. 3 the basic methodology of our approach is discussed. Section 4 presents architecture and protocols of our approach on traditional networks. Section 5 shortly describes the implementation details and test results. In Sect. 6 the evaluation of effectiveness and overhead of our approach is provided, and Sect. 7 concludes the paper.

2. Related Work

Lee and Thing [12] propose a port hopping (PH) technique,

where UDP/TCP port used by the server varies as a function of time and a cryptographic key shared between the server and clients. They implement PH using the socket communications for the UDP protocol, and for setting up TCP communications. Once the TCP connection is established, the TCP protocol will reuse the given port number for the entire connection. The potential problem is that if the TCP connection is long, then it may be vulnerable to attack.

Badishi *et al.* [13], [14] develop a port-based rationing channel (PRC) protocol, which introduces an ack-based method and overcomes the problem on synchronization. PRC performs pseudo-random port hopping, providing an light packet filtering mechanism. Unfortunately, PRC only applies to two-party communication since any other third-party cannot join in the communication without the ack information. Badishi *et al.* [15] further propose an architecture to build dams to protect servers from DoS attacks, which is called Beaver, based on the concept of PRC.

Sifalakis *et al.* [16] utilize network address hopping (NAH) to enhance data protection for packet communication, which uses the accurate time to determine the transmission channels to be selected. The field of communication is divided into several transmission channels, each with a unique pair of IP addresses of source and destination. The drawback is that the disclosure of hopping sequence and the network delay will lead to complete protection failure.

Antonatos *et al.* [5] propose a LAN-level network address space randomization (NASR) scheme based on DHCP update. NASR is not transparent to the end-hosts because DHCP changes are applied to the end-host itself which results in disruption of active connections during address transition. Moreover, NASR provides very limited unpredictability and mutation speed because its IP mutation requires DHCP and host to be reconfigured for this purpose (the maximum IP mutation speed is once every 15 minutes).

Dunlop *et al.* [17] develop the Moving Target IPv6 Defense (MT6D), which takes advantage of IPv6 networks allowing nodes to advertise their own addresses. MT6D encapsulates TCP in UDP to prevent TCP connection disruptions from address rotations, and rotates address using a new computed obscured IPv6 header (i.e. MT6D header). The drawback of MT6D is that it incurs a 62-byte overhead to each MT6D packet, and doesn't apply to rotate IPv4 addresses since the address space is small.

Jafarian *et al.* [18] develop the OpenFlow Random Host Mutation (OF-RHM) architecture, in which the OpenFlow controller frequently assigns each host a random virtual IP selected from the ranges that assigned to the subnet. One limitation of OF-RHM is that a named host can still be reached via DNS, and OF-RHM is not deployable on traditional networks. In order to solve this problem, Al-Shaer *et al.* [19] further present a similar IP mutation technique called Random Host Mutation (RHM). However, the problem of assigning ranges to subnets in OF-RHM and RHM is NP-hard.

Jafarian *et al.* [20] present a spatio-temporal address mutation (STAM) technique which enables host-to-IP bind-

ing of each destination host to vary randomly based on source identity as well as time. Although this scheme is shown to be more effective against various sophisticated reconnaissance attacks, they don't consider port mutation which restricts their application scenarios as well as defense ability against internal attackers.

Atighetchi *et al.* [21] describe two designs for implementing port and address hopping in the APOD (Application that Participate in their Own Defense) project, which uses hopping tunnels and NAT gateway to implement address and port randomization to disguise the identity of end parties from sniffers. However, this work doesn't include the detailed implementation as well as relevant numerical data for the experimental performance results.

Shi *et al.* [22] propose a port and address hopping tactic with a timestamp-based synchronization scheme, which uses a hopping algorithm shared between the server and trusted hosts to implement hopping-based cyber-defense. In order to expand the poor portability and scalability of the tactic in [22], they introduce a full service hopping framework implemented through mobile java agent in [23], and design a spokesman scheme to solve the problem of precise time synchronization. However, they make a too strong assumption that attackers know nothing about the hopping algorithm, and they didn't give any detailed information about the hopping algorithm. Another potential problem is that the spokesman agent or timestamp server may become the vulnerable node instead of the server.

In summary, the existing port hopping methods [12]–[15] constantly mutate service port to a random port selected from the available port space. The address hopping/mutation works [5], [16]–[20] hide service identities by replacing service address with a temporal routable address selected from its subnet. Due to the limitation of the available port and routable address space, both port hopping and address hopping/mutation provide limited unpredictability and defense ability. These techniques are not effective against various sophisticated reconnaissance and coordinated information gathering such as cooperative reconnaissance and advanced persistent threat. Although paper [20] tries to introduce spatial mutation to increase its effectiveness, but do not consider port mutation which restricts its applicability. The port and address hopping works [21]–[23] combines port hopping and address mutation to increase their defense ability by simultaneously hopping port and address. Nevertheless, they do not provide detailed algorithms and implementations, and they usually consider one kind of hopping, i.e., temporal hopping, which restricts its applicability against coordinated attack models.

Different from the existing techniques, RPAH hides client identity by deploying client address mutation, and hides service identity through deploying port and address hopping that has three hopping frequencies (i.e., source hopping, service hopping and temporal hopping). Besides, RPAH introduces port and address demultiplexing and access control mechanisms, which provide a convenient attack detection method as well as the maximum unpredictability

by uniformly rotating address and port to the whole address and port space.

3. Methodology

Our aim is to provide a dynamic tactic that constantly changes a service's identity, i.e., IP address and port. The intention is to both hide the service's real identity and confuse attackers during reconnaissance. In RPAH networks, the real port ($rPort_s$) and real IP address (rIP_s) of a server host are replaced by a short-lived virtual port ($vPort_s$) and a routable virtual IP (vIP_s), and packets are routed based on the vIP_s . The valid $vPort_s$ and vIP_s after authentication are automatically translated to their corresponding $rPort_s$ and rIP_s at the hopping gateways/engines of the destination host.

To be useful, our RPAH must address both IPv4 and IPv6 address schemes. Although the scarcity of IP address makes the unused address space small and highly limited in IPv4 networks, our approach combines IP addresses with port numbers whose range is about 2^{16} to provide an acceptable defense performance. In RPAH, time is divided into discrete intervals I_n , $n = 0, 1, 2, \dots$, each of duration T . A client must use different $vIP_s:vPort_s$ pairs to reach the service in different time intervals. An example of RPAH communication in different time intervals I_1 and I_2 is depicted in Fig. 1, where there are two different service applications S_1 and S_2 running on the same server host. As an example of source hopping, client C_1 reaches service S_1 via $IP_1:Port_1$, while C_2 reaches S_1 via $IP_3:Port_3$ in time interval I_1 . Meanwhile, as an example of service hopping, client C_2 reaches S_2 via $IP_3:Port_4$ during the same time interval I_1 . As an example of temporal hopping, client C_1 reaches service S_1 via $IP_1:Port_1$ during the first time interval I_1 , while reaches S_2 via $IP_2:Port_2$ during the second time interval I_2 .

In order to hide both clients' and servers' real network identities, we perform address mutation for clients and perform port and address hopping for servers. RPAH provides a domain-level address hopping, we limit the server/client address on the server/client domain other than the public addresses because (i) it would explode the size of routing tables, the number of routing updates and the frequency of recomputing routes, (ii) it would result in tremendous administrative overhead for reconfiguring mechanisms that make address-based decisions and (iii) it would require global coordination for being implemented. Therefore,

Fig. 1 The address and port assignment of RPAH

using domain addresses is an applicable choice and the packets after address hopping are still routable without routing updates. In this paper, we define domain C as a server/client network that contains 2^8 addresses, domain B as a server/client network that contains 2^{16} addresses, and domain A as a server/client network that contains 2^{24} addresses, respectively.

3.1 Client Address Mutation

In order to hide clients' real identity, we perform address mutation for clients. All the clients from the same domain are associated with a virtual address range (VAR_c), which represents the whole address space of the client domain. We perform client address mutations for setting up communications. Specifically, we randomly select a virtual IP (vIP_c) from VAR_c to replace the client's real IP (rIP_c). Once the connection is established, the client will reuse the given vIP_c for the entire connection. Since an address can not be used by any two different clients in the same domain during the same time interval, we keep a table that records the usage of VAR_c and exclude the in-using addresses for every new connection.

Let t_s represents the timestamp of the mutation system, which can be obtained by calling a system function (e.g. $time()$). The vIP_c for the client address mutation is generated based on the following function $F_c(\cdot)$ of timestamp t_s and VAR_c :

$$vIP_c = F_c(t_s, VAR_c), \quad (1)$$

Specifically, $F_c(\cdot)$ firstly generates a pseudo-random number based on a pseudo-random function (e.g. $rand()$ in Linux) using timestamp t_s as its seed. Then, the pseudo-random number is further used to determine a virtual address vIP_c from VAR_c . The vIP_c will be stored in the table and kept active and unchanged until the end of the session. Here, the port number of the client kept unchanged because it is randomly assigned and selected by the operating system, and the client's identify information will not be leaked out from this random port number.

3.2 Service Port and Address Hopping

In order to hide servers' real identities, each server host is associated with a virtual address range (VAR_s), which is the whole address space of the server domain, e.g., VAR_s contains 254 valid IP addresses for a C domain that contains 2^8 addresses, since 255 is reserved for broadcast and 0 is not a valid host number. Each service application running on the server host is associated with a virtual port range (VPR_s), which contains 65535 ports from port number 1 to 65535, since 0 is a reserved port number.

3.2.1 Address Encoding/Decoding

In RPAH, let T represents the time interval length that defines the lifetime of $vIP_s/vPort_s$, and t_i represents the current

Algorithm 1 address encoding algorithm

Input:

- 1: the hopping period T
- 2: the shared secret key K_a
- 3: vIP_{ct} , rIP_s and rIP_{st}

Output:

- 4: generate n_r of the current time interval
 $n_r = HF(K_a, \lfloor t_i/T \rfloor)$ $n_{rl} = n_r/2^k$ $n_{rr} = n_r \% 2^k$
- 5: $vIP'_{ct} = vIP_{ct} \oplus n_{rl}$
- 6: $rIP'_{st} = rIP_{st} \oplus n_{rr}$
- 7: $vIP_{st} = vIP'_{ct} \oplus rIP'_{st}$
- 8: **if** $((vIP_{st} == 0) \parallel (vIP_{st} == 2^k - 1))$ **then**
- 9: $vIP_{st} = vIP_{st} \oplus 1$
- 10: **end if**
- 11: $vIP_s = (rIP_s \& (2^{32} - 2^k)) \parallel vIP_{st}$

timestamp of hopping system that can be obtained by calling a system function (e.g. $time()$), then the time parameter used for hopping is defined as $\lfloor t_i/T \rfloor$. In a new time interval, a $vIP_s:vPort_s$ pair is generated based on a function $F_s(\cdot)$ of secret key K , source identity ($srcID$), service identity ($svcID$) and time parameter $\lfloor t_i/T \rfloor$ as follows:

$$vIP_s = F_s(K_a, vIP_c, rIP_s, \lfloor t_i/T \rfloor), \quad (2)$$

$$vPort_s = F_s(K_p, rPort_c, rPort_s, \lfloor t_i/T \rfloor), \quad (3)$$

Here, $F_s(\cdot)$ is a formal description of service port and address hopping, whose detailed processes are described in Algorithm 1 and 2. K_a and K_p represent the secret key of address hopping and port hopping respectively, and they are different parts of the same secret key K shared between authenticated clients and the server. Key agreement can be achieved by existing symmetric key cipher or public key cryptography (PKC) described in [12]. In this paper, we assume that all authenticated clients have already got their keys in a secure way.

The VAR_s of the server host is the whole address space of the server domain, which means only the host part other than the network part of a server IP address should be encoded. As an example, for a k -bit length host part address hopping, only the last k -bit of a 32-bit server IP address should be encoded. Let IP with subscript t represent the k -bit host part address (e.g. IP_{ct} represents host part of a client address). The detailed description of Eq. (2) based address encoding process is shown in Algorithm 1, where $HF(\cdot)$ is a one-way keyed hash function used to generated a $2k$ -bit random number n_r based on secret key K_a and the current time t_i . Further, n_r is divided into a k -bit left part n_{rl} and a k -bit right part n_{rr} , which are used to perform exclusive OR operations with vIP_{ct} and rIP_{st} and generate the hopping address vIP_s . Invalid address whose IP_{st} is 0 or broadcast address (i.e. $2^k - 1$) may be included in the address hopping, to cope with this, we designed a generic process by performing exclusive OR operation with 1 (i.e., step 9) to make sure that the generated vIP_{st} falls into the VAR_s . Note that, the number 1 can also be any other number from 1 to $2^k - 2$ whose last bit is 1.

The address decoding is the reverse process of the address encoding, in which we use vIP_{ct} and vIP_{st} to decode

Algorithm 2 address decoding algorithm**Input:**

- 1: the hopping period T
- 2: the shared secret key K_a
- 3: vIP_{ct} , vIP_s and vIP_{st}

Output:

- 4: generate n_r of the current time interval
 $n_r = HF(K_a, \lfloor t_i/T \rfloor)$ $n_{rl} = n_r/2^k$ $n_{rr} = n_r \% 2^k$
- 5: $vIP'_{ct} = vIP_{ct} \oplus n_{rl}$
- 6: $rIP_{st} = vIP'_{ct} \oplus vIP_{st} \oplus n_{rr}$
- 7: **if** $((vIP_{st} == 1) \parallel (vIP_{st} == 2^k - 2))$ **then**
- 8: $rIP'_{st} = rIP_{st} \oplus 1$
- 9: **end if**
- 10: $vIP'_{st} = vIP'_{ct} \oplus rIP'_{st} \oplus n_{rr}$
- 11: **if** $vIP'_{st} == vIP_{st}$ **then**
- 12: $rIP_s = (vIP_s \& (2^{32} - 2^k)) \parallel rIP'_{st}$
- 13: **else**
- 14: $rIP_s = (vIP_s \& (2^{32} - 2^k)) \parallel rIP_{st}$
- 15: **end if**

the real service IP address (rIP_s). The address decoding process is shown in Algorithm 2: i) It generates n_r based on $HF(\cdot)$. ii) It uses n_r to perform exclusive OR operations with vIP_{ct} and vIP_{st} and generates the host part (rIP_{st}) of the real service IP address (rIP_s). iii) It judges whether vIP_{st} is 1 or $2^k - 2$ (note that this branch condition varies with the number used in step 9 of Algorithm 1), if so, recomputes another possible rIP'_{st} . iv) It determines the valid host part from rIP_{st} and rIP'_{st} and figures out the real service address rIP_s based on special treatment from steps 10 to 15.

Note that, because the random number n_r used in address encoding and decoding is generated based on a one-way keyed hash function $HF(\cdot)$ of secret key K and the current time $\lfloor t_i/T \rfloor$ of the hopping system, the attackers cannot generate the valid addresses without the secret key K even though they know the detail information and algorithms of our RPAH. The port encoding and decoding can be performed in the same way as address encoding and decoding, the only difference is that the whole 16-bit port number should be encoded, no more tautology here.

The main goal of RPAH is to maximize hopping unpredictability. In this paper, service port and address hopping is triggered by three different parameters: source identity, service identity and time, which provides RPAH with three hopping frequencies, i.e., source hopping, service hopping and temporal hopping.

3.2.2 Source Hopping

Source hopping is service port and address hopping based on $srcID$, RPAH uses vIP_c as the $srcID$ of service address hopping (i.e. Eq. (2)) and use $rPort_c$ as the $srcID$ of service port hopping (i.e. Eq. (3)), respectively. Source hopping is performed by associating the $srcID$ to each port and address hopping. Consequently, various clients must use different $vIP_s:vPort_s$ pairs to reach service S_i running on a server host, which means $vIP_s:vPort_s$ generated based on client C_i 's identity cannot be used by any other client to connect the same service S_i during the same time interval.

3.2.3 Service Hopping

Service hopping is port and address hopping based on $svcID$, RPAH uses rIP_s as the $svcID$ of service address hopping (i.e. Eq. (2)) and use $rPort_s$ as the $svcID$ of service port hopping (i.e. Eq. (3)), respectively. Consequently, clients must use different vIP_s to reach various server hosts, and use different $vPort_s$ to reach various service applications running on the same server host. Meanwhile, under the control of port and address demultiplexing and access control mechanisms expounded in the later part of this paper, vIP_s generated based on server host H_i 's $svcID$ can only be used to reach server H_i , and $vPort_s$ generated based on service S_i 's $svcID$ can only be used to reach service S_i .

3.2.4 Temporal Hopping

Temporal hopping is port and address hopping based on time, the time of the hopping systems are synchronized by a standard Internet time server (e.g. time.nist.gov) using network time protocol (NTP). More specifically, after time synchronization, current time t_i of the hopping system obtained by calling a system function (e.g. `time()`) is a nonzero integer, and the time parameter used for temporal hopping is obtained by adding a rounding operation to t_i on time period T (i.e. $\lfloor t_i/T \rfloor$), incremented once per hopping interval. RPAH uses $\lfloor t_i/T \rfloor$ as the time parameter of service port and address hopping. Consequently, the valid $vIP_s:vPort_s$ changes once per hopping interval, and clients must use different $vIP_s:vPort_s$ pairs to reach a service application running on a server host in different time intervals. Determining the duration of T is a trade-off between performance and robustness. Short T will result in high overhead of packet processing and synchronization, while longer T decreases the effectiveness of thwarting attacks.

4. RPAH: Architecture, Protocol and Support Mechanisms

In this section, we discuss the architecture of RPAH, communication protocols and its support mechanisms.

4.1 Architecture

The architecture of RPAH is depicted in Fig. 2, in which the moving target (MT) and non-moving target (non-MT) servers/clients represent servers/clients with and without port and address hopping, respectively. The deployment is accomplished via three components: hopping gateway of client domain (HG_c), port hopping engine of server host (PHG) and hopping gateway of server domain (HG_s). HG_s is located at the network edge of the MT servers (e.g. on the gateway in front of subnet switch of the server domain) and performs address authentication and translation according to address decoding and access control algorithms. PHG is

Fig. 2 The architecture of RPAH

located on the MT server host directly and performs port authentication and translation according to port decoding and access control algorithms. In order to be transparent to client, HG_c is located at the network edge of the MT clients (e.g. on the gateway in front of subnet switch of the client domain) and performs client address mutation and mapping, service port and address hopping and translation according to their separate algorithms.

HG_c is the component that performs client address mutation and service port and address hopping for clients, and it has various functions. Firstly, it performs service port and address hopping by generating the current $vIP_s:vPort_s$ of the MT server based on Eq. (2) and Eq. (3) to guarantee unpredictability. Secondly, it randomly selects vIP_c from VAR_c of client hosts based on Eq. (1), and replaces rIP_c with vIP_c to hide the real identities of client hosts. Thirdly, it performs address replacements for DNS responses from DNS server, such that clients receive the server's current vIP_s as the destination address and the rIP_s of the server host is untouched. Lastly, it performs port and address translation for clients transparently without any changes of client hosts. It maintains rIP_c-vIP_c , rIP_s-vIP_s and $rPort_s-vPort_s$ mapping tables and performs client address mutation and mapping and service port and address hopping and translation for egress and ingress packets. The general client address mutation and service port and address hopping algorithm is presented in Algorithm 3.

HG_s is the component that performs address authentication and translation for MT server hosts, and it has various functions. Firstly, it extracts vIP_c and vIP_s of the incoming packet and decodes its rIP_s based on Algorithm 2. Secondly, HG_s performs address authentication and access control by checking if rIP_s is valid and active in the server domain, if so, HG_s accepts the packet and updates the address mapping table; if not, which means the packet is using invalid address and it will be denied and dropped. Thirdly, it maintains a vIP_s-rIP_s mapping table and performs address translation for incoming and outgoing packets. Specifically, it translates destination vIP_s to rIP_s for ingress packets, and replace source rIP_s with vIP_s for egress packets. The general address authentication and translation algorithm is presented in Algorithm 4.

Algorithm 3 client address mutation and service port and address hopping algorithm

```

1: for all egress packet  $p$  from a client to a server do
2:   if  $p$  is a new connection setting up packet then
3:     generate  $vIP_c$  based on Eq. (1)
4:     store  $rIP_c - vIP_c$  mapping item in hash table
5:     replace  $rIP_c$  with  $vIP_c$ 
6:   else
7:     search out  $vIP_c$  from hash table
8:     replace  $rIP_c$  with  $vIP_c$ 
9:   end if
10:  generate  $vIP_s : vPort_s$  based on Algorithm 1
11:  store  $rIP_s - vIP_s$ ,  $rPort_s - vPort_s$  mapping items in hash table
12:  replace  $rIP_s$  with  $vIP_s$ ; replace  $rPort_s$  with  $vPort_s$ 
13:  send  $p$  to the network after re-checksum
14: end for
15: for all ingress packet  $p$  from a server to a client do
16:   search out  $rIP_s$ ,  $rPort_s$  and  $rIP_c$  from hash tables
17:   translate  $vIP_c$  to  $rIP_c$ , translate  $vIP_s$  to  $rIP_s$  and  $vPort_s$  to  $rPort_s$ 
18:   send  $p$  to the client host after re-checksum
19: end for

```

Algorithm 4 address authentication and translation algorithm

```

1: for all ingress packet  $p$  from a client to a server do
2:   decode  $rIP_s$  based on Algorithm 2
3:   match  $rIP_s$  based on hash table
4:   if  $rIP_s$  is valid and active in the server domain then
5:     translate  $vIP_s$  to  $rIP_s$ 
6:     send  $p$  to the server after re-checksum
7:   else
8:     drop  $p$ 
9:   end if
10: end for
11: for all egress packet  $p$  from a server to a client do
12:   search out  $vIP_s$  from hash tables
13:   replace  $rIP_s$  with  $vIP_s$ 
14:   send  $p$  to the client host after re-checksum
15: end for

```

PHG is the component that performs port authentication and translation for MT server hosts, and it has various functions. Firstly, it extracts $rPort_c$ and $vPort_s$ of the incoming packet and decodes its $rPort_s$ based on port decoding algorithm. Secondly, PHG performs port authentication and access control by checking if $rPort_s$ is valid and active in the server host, if so, PHG accepts the packet and updates the port hash chain table; if not, which means the packet is using invalid port number and it will be denied and dropped. Thirdly, it maintains a $vPort_s-rPort_s$ mapping table and performs port translation for incoming and outgoing packets. Specifically, it translates destination $vPort_s$ to $rPort_s$ for ingress packets, and replaces source $rPort_s$ with $vPort_s$ for egress packets. The port authentication and translation algorithm is similar as Algorithm 4, no more tautology here.

4.2 Protocol

There are two ways to communicate with a MT server: using its domain name and real IP address rIP_s . These two

Fig. 3 Communication via name

Fig. 4 Communication via rIP_s

scenarios of communication are depicted in Figs. 3 and 4.

Figure 3 shows that when a DNS query is sent by a client host to resolve the domain name of a MT server host (e.g. HTTP Server), after verification of the client's legitimacy, the DNS response is intercepted by the HG_c and the rIP_s of the MT server is replaced with a new generated vIP_s . As a result, the MT client receives vIP_s mapping to MT server name and initiate its connection accordingly. In the communication packets the well-known service port $rPort_s$ (i.e. 80) is used as the destination port which is replaced by a new generated $vPort_s$ when they pass through the HG_c . Meanwhile, the source rIP_c of the outgoing packet is replaced by a random selected vIP_c .

Figure 4 shows how a client can reach a MT server host using the real IP address rIP_s of the MT server (e.g.

Fig. 5 IP and port demultiplexing

FTP Server). In this scenario, the client initiates connection using rIP_s and $rPort_s$, when HG_c receives the packet, it requests and authorizes access for this client. If access is granted, HG_c will replace the destination address rIP_s with a new generated vIP_s and send the packet to the network. Meanwhile, the packet is intercepted by the HG_c and the $rPort_s$ is replaced with a new generated $vPort_s$. It is important to note that this authorization is performed once per TCP connection and once per UDP packet includes rIP_s as destination.

In RPAH networks, routings are restricted to vIP_s of the destination hosts and service requests are restricted to $vPort_s$ of the service applications in order to ensure the effectiveness of port and address hopping.

4.3 Support Mechanisms

In this section, we discuss the main support mechanisms in RPAH for the port and address hopping enforcement, i.e., IP and port demultiplexing, access control and attack detection mechanisms.

4.3.1 IP and Port Demultiplexing

As the result of source hopping, different clients must use various vIP_s and $vPort_s$ to reach the same server host and service application, but it is possible that several clients use the same vIP_s to connect different servers running in the same server domain or use the same $vPort_s$ to connect different services running on the same server host during the same time interval. This is a one-to-many relationship and requires demultiplexing.

The model of IP and port demultiplexing is depicted in Fig. 5. The server domain may receive packets with the same destination address vIP_s from various clients to different servers. After address decoding and translation, their separate addresses rIP_s are decoded from the same virtual destination address vIP_s , and the packets are delivered to their respective servers based on the rIP_s . Likewise, the server host may also receive packets with the same destination port number $vPort_s$ from different clients to various service applications. After port decoding and translation, their separate port numbers $rPort_s$ are decoded from the same virtual destination port number $vPort_s$, and

Fig. 6 An example of address access control

the packets are delivered to their right services based on the $rPort_s$.

IP and port demultiplexing mechanism provides RPAH with a better randomness and unpredictability since it makes all ports/addresses of VPR_s/VAR_s reusable for various servers and service applications. Therefore, the range constraints of virtual address and port in RPAH is the whole domain address space and the entire communication port space.

4.3.2 Access Control and Attack Detection

Access control [24] is used to determine the validity of each vIP_s and $vPort_s$ used by a client to connect the server during communications. Every time a connection setting up packet from a client arrives at the hopping gateway of the server domain (HG_s), after extraction of its source address vIP_c , source port $rPort_c$ and destination address vIP_s , then the rIP_s is decoded based on Algorithm 2. If a server host whose IP address equals to rIP_s is active in the server domain, which means the connection request passes the address authentication and the packet will be forwarded to the right server after address translation. Meanwhile, the connection state will be stored into the mapping table. Thereafter, we perform address decoding and authentication for the first packet of every new arrival connection based on Algorithm 2 in a hopping interval, while perform address matching and translation based on mapping table for subsequent packets arrived in the same time interval.

Figure 6 illustrates how the hopping gateway HG_s processes received packets. HG_s keeps a mapping hash chain table HT_s based on a common hash function $hf(\cdot)$ of vIP_c and $rPort_c$. Specifically, $hf(\cdot)$ receives a $vIP_c:rPort_c$ pair as its input and generates an integer i as its output, and i is used as the index number of hash chain list of the vIP_s -to- rIP_s mappings. In a new hopping interval, every time a new packet arrives, after decoding of its rIP_s and address authentication, a two-tuple record ($vIP_c:rIP_s$) is stored in HT_s based on the output of hf . Thereafter a packet arrives, HG_s checks if the output of $hf(vIP_c, rPort_c)$ matches HT_s . If so, it checks if vIP_s extracted from the packet is equal to the vIP_s of the table item, if so, means the packet is legal and its vIP_s will be directly translated to rIP_s and the packet will be forwarded to the right server host. Otherwise, HG_s decodes the rIP_s of the current packet based on Algorithm 2, and checks if the rIP_s is valid in the server domain, if so,

it updates HT_s and adds a new table item in HT_s and the packet will be forward to the right server, if not, the packet will be denied and dropped. An example of address-based access control is shown in Fig. 6, where the packet destined to IP_2 using source address IP_{c2} and source port $Port_{c2}$, while the decoded destination address rIP_s is invalid in the server domain will be dropped as not fulfilling the address hopping algorithm, and the packet destined to IP_{s3} which is the real address of an active server host will be dropped because the real address IP_{s3} cannot be directly used to connect the server host.

HG_s records the time information of each table item, and delete it after two hopping intervals considering the network transmission delay. Note that, only the packet whose vIP_s fulfils Eq. (2) based address hopping and the rIP_s decoded point to a valid and active server host in the server domain can be stored in HT_s , so little memory is required to keep these items. Access control of the communication port that performed on the server host is similar to the address access control, no more tautology here.

Compared to the traditional security mechanisms such as IDS/IPS and firewall in legacy networks that using predefined rules to protect the static service model, which can only detect attacks known already. RPAH can provide us with a more convenient and effective attack detection mechanism, since all connection attempts using static, invalid or out-of-date addresses and/or ports that mismatch the access control hash table HT_s can be easily detected and controlled. We can record states and frequencies of these failure connection attempts, analysis and recognize their attack behaviors. Further, we can take actions (e.g., deny/discard, redirect to honey net or provide fake service) to cope with these attacks.

5. Implementation and Test

To study and demonstrate the feasibility of RPAH, we implemented RPAH in an experimental network in our university campus network. The network contains two C domains. A client domain that contains client hosts and a server domain that contains an *Apache* Web server host, a FTP server host and a TFTP server host. VAR_s contains 254 addresses and VPR_s contains 65535 ports. The HG_c , HG_s and PHG components were implemented on Ubuntu Linux (kernel 3.17.3) using netfilter framework [25]. We added function handles into netfilter's *PRE_ROUTING* and *POST_ROUTING* chains to hook incoming and outgoing packets, and performed port and address hopping, authentication, translation and access control with a fixed time period (e.g. 5 seconds).

As an example of RPAH communication, a link topology of two FTP data uploading connections was depicted in Fig. 7, each connection of duration one minute. The client (the host part of its IP address rIP_c is 10) uploads a file to the server host (the host part of its IP address rIP_s is 20 and data port number $rPort_s=20$). The topology is

Fig. 7 The link topology of two FTP connections

Fig. 8 The address usage probability in hopping

much more complicated than the static communications in the legacy network depicted by the dashed line from the client to the server. In each connection, a vIP_c (i.e. 110 in the first connection and 189 in the second connection) was used to replace the rIP_c (10) in the client address mutation and 12 different $vPort_s:vIP_s$ pairs were used to replace the $rPort_s:rIP_s$ (20:20) in the service port and address hopping.

The main goal of RPAH is to maximize hopping unpredictability. A good hopping algorithm should encode address/port uniformly into the whole address/port space, which means every address/port in the space should be used with equal probability. To achieve this, we introduce a one-way hash function in RPAH for address/port encoding and decoding, which generates a pseudo-random number for each hopping to make sure the hopping unpredictability. To evaluate this, we generated 10 thousands hopping addresses and the statistics result is shown in Fig. 8, where the dashed line represents the probability density function of the address uniform distribution (i.e. $1/VAR_s \approx 0.39\%$). We can see that the actual usage probability of each hopping address is approximate to 0.39%, which means the addresses used for hopping fulfils a uniform distribution. Therefore, RPAH can rotate a service IP to a random one uniformly selected from the whole address pool, which provide the maximal hopping unpredictability.

In order to evaluate the influence to services introduced by RPAH, we perform various network activities during port

and address hopping including web browsing, downloading files, and these network connections were not affected or interrupted even persist for more than 10 minutes. We also use *ApacheBench* [26] to perform concurrent connection tests. Specifically, we use *ApacheBench* to perform 20 tests of 1000 web requests concurrences up to 50, and the connection success rates are 100% for all the tests. Our implementation simply proved that the RPAH is a feasible port and address hopping approach. A more detailed and comprehensive evaluation of RPAH will be presented in the next section.

6. Evaluation

In the previous section, we conducted a simple test for RPAH and demonstrated its feasibility. In this section, we assess the overall effectiveness of RPAH in resisting network reconnaissances through theoretic analysis, simulation and experiments on our campus network. Furthermore, we also evaluate the overhead of RPAH through experiments on *PlanetLab* [27] platform.

6.1 Attacker Model

In our evaluation, we consider the attackers as non-adaptive attackers, and they are not able to adapt to our RPAH mechanism without the shared secret key. For the sake of gathering information of the target systems, the attackers will use random port numbers and IP addresses to perform host discovery and service scanning in the first step of launching network attacks (i.e. network reconnaissance).

6.2 Effectiveness of RPAH Resists Random Scanners

Random scanner is the primary instrument for network reconnaissance. The attackers usually use scanners such as *Nmap*, *Superscan* [28] to discover active server hosts and further identify available services running on the target host. More advanced scanners (e.g. *Nessus* [29]) are usually used to gather useful information such as OS types, software versions and vulnerabilities of the target systems, and these information are useful for attackers to launch appropriate attacks. Depending on whether the scanning are performed by authorized users know about the secret key or unauthorized users without secret key, random scanning can be divided into two categories. The first category is random scanning performed by authorized users with secret key. In this case, legal host H_i uses scanners to gather information of the target network, but the information gathering is limited to vIP_s and $vPort_s$ active in that time interval. These information can only be used by host H_i and they will become out-of-date in the next time interval. Besides, these information are useless to any other host H_j because of source hopping. The second category is random scanning performed by unauthorized users without secret key.

In this section, we focus on the latter category of random scanning such as host discovery and port scanning

to evaluate the effectiveness of RPAH against scanners. Usually, host discovery to a single server host or port scanning to a single service application can be achieved through sending a single packet to the target host/service (e.g., using a ICMP packet to identify whether a server host is active or not, or using a SYN packet to test whether a server port is open or not). Therefore, in this paper, we define the number of scans N as the total number of probing packets sent out by the scanner, each probing packet here represents a scan event. Although a single scan event in a practical attack may be composed of multiple packets, several port and address hopping events may happen during the scanning and disrupt the reconnaissance process; therefore the practical scan success rate will be far below the evaluation result of host discovery and port scanning derived in this paper.

Consider a scenario as follows: given a scanner host in a client domain and a MT server host in a server domain of an enterprise network. Let V_{Ac} , V_{Pc} , V_{As} and V_{Ps} represent the size of the client's address space VAR_c and client's port space VPR_c , the size of the server's address space VAR_s and server's port space VPR_s , respectively. The valid vIP_s and $vPort_s$ of the server host change once per hopping. In this scenario, the scanner seeks to scan the MT server host while the server host tries to hide its real service address and port through RPAH. Suppose there is only one service running on the MT server host. Let scanning rate η represents the number of probing packets that a scanner host sends out per second and f represents the hopping rate. Then, the scanning capacity R is defined as

$$R = \frac{\eta}{f}, \quad (4)$$

which represents the number of scans performed by the scanner per hopping. For the sake of information gathering, the best choice for the attacker is performing repeat scanning on the address and port space scanned before after each hopping, and performing non-repeat scanning in every hopping interval. Generally, some security systems would be deployed to protect the server host; therefore the number of scans N the attacker can perform to reach the server host is limited. In this scenario, given N scans ($N \leq R$), the scanner can scan the whole space in each hopping interval through non-repeat scanning. For the i -th probing packet the scanner sends out, the probability of discovering the target is

$$p_i = \frac{1}{V-i}.$$

Therefore, given n scans ($n \leq N$), the scanner will miss the target with probability

$$P_n = \prod_{i=0}^{n-1} \left(1 - \frac{1}{V-i}\right). \quad (5)$$

Accordingly, given N scans ($N > R$), the scanner cannot scan the whole space in a hopping interval; consequently the scanner can only scan R addresses/ports in each hopping

Fig. 9 The scan success rate vs. the number of scans for different scanning capacities ($R = 1 \rightarrow V$).

interval. For N scans, there are $\lfloor \frac{N}{R} \rfloor$ hopping intervals and the remainder number of scans is $N - \lfloor \frac{N}{R} \rfloor * R$. For $\lfloor \frac{N}{R} \rfloor$ hopping intervals, the scanner will miss the target with probability

$$P_R^{\lfloor \frac{N}{R} \rfloor} = \left(\prod_{i=0}^{R-1} \left(1 - \frac{1}{V-i}\right) \right)^{\lfloor \frac{N}{R} \rfloor}. \quad (6)$$

Based on Eqs. (5) and (6) we can derive that given N scans the scanner can discover the target server host with probability

$$P = \begin{cases} 1 - \prod_{i=0}^{N-1} \left(1 - \frac{1}{V-i}\right) & N \leq R, \\ 1 - \left(\prod_{i=0}^{R-1} \left(1 - \frac{1}{V-i}\right) \right)^{\lfloor \frac{N}{R} \rfloor} * \prod_{j=0}^{N-\lfloor \frac{N}{R} \rfloor * R-1} \left(1 - \frac{1}{V-j}\right) & N > R. \end{cases} \quad (7)$$

where $V = V_{Ac} \cdot V_{As}$ represents the number of vIP_c - vIP_s cases for the scanner to reach the server host for host discovery in the reconnaissance phase. Further, the scanner may perform port scanning to discover and identify services running on the target server host, and it will discover the target service with the probability shown in Eq. (7) where $V = V_{Ac} \cdot V_{Pc} \cdot V_{As} \cdot V_{Ps}$ is the number of vIP_c : $rPort_c$ - vIP_s : $vPort_s$ cases for the scanner to connect the target service.

Figure 9 shows the theoretical and simulated scan success rate for a random scanner in a C domain to discover the target server host in another C domain (here, $V = V_{Ac} \cdot V_{As} = 2^{16}$) with the number of scans N from 0 to 65536 and different scanning capacities $R = 1, 0.2V, 0.4V, 0.6V, 0.8V$ and V . Obviously, the scan success rate increases as the number of scans increases and it reaches 100% when $R=V$ and $N=V$, which means the scanner can search the whole address space per hopping and the whole address space can be scanned. The scan success rate decreases as the scanner's scanning capacity decreases and it falls to 63% when $R=1$, this is an extreme case known as perfect hopping [30], where a hopping event happens every time a probe attempts. In this

Fig. 10 The scan success rate vs. the number of scans for different size of networks ($R = 1$).

scenario, the scan success rate defined by Eq. 7 becomes

$$P = 1 - \left(1 - \frac{1}{V}\right)^N, \quad (8)$$

which was depicted in Fig. 9 using line with square symbols. The scan success rate increases as the number of scans increases and ultimately to $1 - e^{-1} \approx 0.63$ in the worst-case scenario when the whole address/port space can be scanned ($N=V$), which means the scanner will miss 37% portion of the target hosts.

Perfect hopping ($R = 1$) is the safest port and address hopping scheme since it provides the maximum hopping frequency a hopping system can reach. Perfect hopping is a defense space maximization model, which forces the attacker to scan the whole port and address space after each hopping. As shown in Fig. 9, the performance of $R = 1$ is not far from the performance of $R = 0.2V$. Generally, some security systems would be deployed to protect the server host, and the scanning rate η an attacker can perform in the attack is limited. Therefore, we can infer that $R = \eta T \ll V$. Without consideration of the implement overhead, perfect hopping is the guiding principle of setting the hopping frequency of our RPAH, combine with the concrete attack scenarios and defense abilities of the deployed security systems. Therefore, we can use the scenario of perfect hopping ($R=1$) to approximately evaluate the effectiveness of our RPAH. In this section, we use perfect hopping ($R=1$) as an example to evaluate the influence of different size of networks to the scan success rate, where a C-C scanning that represents a scanner in a C domain performing scanning to another C domain ($V \approx 2^{16}$), a C-B scanning that represents a scanner in a C domain performing scanning to a B domain ($V \approx 2^{24}$) and a B-B scanning that represents a scanner in a B domain performing scanning to another B domain ($V \approx 2^{32}$) were depicted as comparisons. The scan success rates are shown in Fig. 10, when $N = 65536$, the scan success rate of the C-C scanning is 63%, while the probability of the C-B scanning is 0.39% and the probability of the B-B scanning to is about 0. The simulation results proved that the scan success rates decrease as the increase of the address space size, and RPAH can save about 37% of vulnerable hosts even though the address space is as small as $V \approx 2^{16}$ and the whole space can be scanned.

Fig. 11 The scan success rate vs. the number of scans for different address hopping techniques.

Figures 9 and 10 show that for a MTD network with RPAH, as the number of reachable scans (N) decreases, scan success rate decreases while deception increases; therefore desired defense performance can be achieved by limiting the number of scans an attacker can perform. For example, if $N \leq 0.2V$, the scan success rate will be below 12%, means at least 88% of network hosts will evade reconnaissance. In RPAH, we designed the access control and attack detection mechanism in Sect. 4.3.2 to limit the number of scans performed by attackers, where multiple scans that disobey the RPAH mechanism could be easily detected and its following scans would be limited or denied. Moreover, packet filtering and rate limiting are two basic functions of conventional security systems such as fire wall, intrusion detection/prevention systems; therefore these security systems can also be deployed to protect the server network and limit the number of reachable scans.

To a C-C scanning, the scanning capacity $R=0.2V$ is approximately equals to 13.1 thousand packets per hopping (i.e., 2.6 kilo packets-per-second (Kpps) when hopping interval $T=5$ seconds), which is beyond the scanning capacity of the most used scanner *Nmap* under the stealthy reconnaissance requirements. To a C-B scanning, although a scanner whose scanning capacity is as high as V can perform a million scans to the target network, its scan success rate is far below 10%, accurately is 5.79% based on Eq. (7). In this case, even though we use the most powerful scanner *ZMap* [31] that can send packets close to the theoretical limit of an Ethernet card, reaching approximately 1.5 million packets-per-second (Mpps) using a 1gbps network card, we can infer that $T \leq 11.18$ seconds based on another equivalent equation of Eq. (4), i.e., $R = \eta \cdot T \leq V$. Therefore, we use hopping interval $T=5$ seconds (i.e. hopping frequency $f=0.2$) as a secure hopping parameter of our RPAH to conduct the following simulations and experiments.

In order to make comparisons between our RPAH and some other hopping techniques as well as the static legacy networks, a random scanner with scanning capacities $R=0.2V$ in a C domain was used to perform host discovery and port scanning to another C domain, and the results are shown in Figs. 11 and 12, respectively. Note that, the port scanning was performed to the target host discovered in the process of host discovery. Since the nature of existing

Fig. 12 The scan success rate vs. the number of scans for different port hopping techniques.

port hopping [12], address hopping [5], [16] and port and address hopping/mutation [18], [19], [21]–[23] schemes is periodic port or/and address rotation. Assume there is only one server host running in the target network, existing works can be simplified as address hopping using the whole domain addresses and port hopping using the whole port space despite their detail implementation techniques, and they were depicted in Figs. 11 and 12 using red line with circle symbols. The results show that our RPAH can provide hundreds times the address diversity and thousands times the port diversity compared to the existing hopping schemes, and the scan success rate of RPAH reaches 63%, while the rate of static legacy network is 100% when the whole address/port space can be scanned. The STAM [20] mutate addresses based on the source identity as well as time, its defense performance of address hopping is the same as our RPAH scheme, while its performance of port hopping is the same as the static network since it do not consider port hopping.

In the practical experiment, we performed 50 times address and port scanning using *Nmap*, no one of the MT server hosts or services were scanned out, and the result of port scanning showed that the scanning packets sent to the MT servers were filtered. The above theoretical analysis and experiment shows that our RPAH is effective in resisting random scanners.

6.3 Effectiveness of RPAH Resists Worms

Worm attacking techniques are more advanced today, which implement various scanning strategies such as uniform scan, hit-list scan, cooperative scan, divide-and-conquer scan, etc. In this section, we study the defense effectiveness of RPAH on various scanning-based worms using worm propagation speeds with and without RPAH.

In RPAH network, the vIP_s and $vPort_s$ of a MT server host vary constantly and the rIP_s and $rPort_s$ are out of touch; therefore effective scanning strategies used in traditional networks do not function well in MTD networks. Specifically, as the result of source hopping, the effectiveness of hit-list scan, cooperative scan and divide-and-conquer scan are badly limited since the vulnerable host ad-

Fig. 13 The number of infected hosts vs. time

dresses and ports obtained by a host is of no use to another. As the result of temporal hopping, a new $vIP_s:vPort_s$ pair will be randomly selected from the whole address and port space in each hopping interval, and the usage probabilities of vIP_s and $vPort_s$ fulfil uniform distributions; therefore the addresses and ports scanned before should be re-scanned after every hopping. In this scenario, divide-and-conquer scan, sequential scan and uniform scan are equivalent in terms of their worm's propagation speeds [32]. Therefore, we focus on uniform scan worms other than non-uniform scan worms because the latter will miss a large part of the active MT hosts.

A uniform scan worm uniformly picks IP addresses in its scanning space to scan and compromise. Suppose a worm has an average scanning rate $\bar{\eta}$, which is the average number of scans an infected host sends out per second. $I(t)$ is the number of infectious hosts at time t and M is the total number of susceptible hosts in the target network. The worm uniformly scans the IP space that has V addresses. Based on our analysis of scanner in Eqs. (7) and (8), the scan worm will miss a portion of the vulnerable hosts (i.e. P_{mis}), and this can be considered to be equivalent to that the whole target network only contains $(1 - P_{mis})M$ vulnerable hosts. Based on [32], the propagation speed of the uniform scan worm in RPAH network can be modeled as:

$$\frac{dI(t)}{dt} = \frac{\bar{\eta}}{V} I(t) [(1 - P_{mis})M - I(t)] \quad (9)$$

In this section, we model the well-known uniform scan worms, Slammer [32]. We use the same parameters for Slammer as what used in [33], i.e., $\bar{\eta} = 4000$ scans/second, $I(0) = 10$. Figure 13 shows the effectiveness of RPAH on propagation of the Slammer worm in a A domain (2^{24} total addresses) with 10^4 vulnerable hosts and various percentages of reachable hosts (α). We can see that with RPAH, worm propagation takes about 100 times more than usual time and takes several decuple times more than using existing methods such as STAM, OF-RHM and RHM. This means that RPAH can slow down the worm propagation significantly. Figure 13 also shows the theoretical number of infected hosts decreases as the percentage of reachable

Table 1 PlanetLab nodes

Country	IP	Country	IP
France	132.227.62.25	China	166.111.4.100
Denmark	141.20.103.211	Finland	193.166.167.5
Britain	200.17.202.195	Japan	133.68.253.243
Russia	93.180.45.10	America	129.10.116.81
Korea	147.46.240.156	Singapore	137.132.80.57

Table 2 Throughput overhead

	1	2	3	4	5
Static (Mbps)	42.36	62.56	91.63	69.38	84.89
RPAH (Mbps)	35.43	54.77	77.05	58.52	75.37
Overhead (%)	16.36	12.45	15.98	15.65	11.21

hosts (α) decreases. Accordingly, 72% of network hosts will evade infection when $\alpha = 1/3$, and 51% of network hosts will evade infection when $\alpha = 2/3$. RPAH can save about 37% of vulnerable hosts in the worst case-scenario when the worm can scan the whole address space ($\alpha = 1$).

6.4 Overhead

The experimental results presented above indicate that using RPAH would be effective in defense, but implementing RPAH would also increase the overhead in the communications. In this section, we use actual implementation on PlanetLab to measure the overhead introduced by RPAH. Note that, since we have no access to the gateway of servers deployed in PlanetLab network, we cannot change the IP addresses of the remote PlanetLab nodes. In order to approximately measure the overhead introduced by RPAH, we perform the entire processes of address and port encoding and decoding, but actually we only change the server's port in the communications.

In the experiment, 10 geographically distinct PlanetLab nodes which are listed in Table 1 around the world were selected to form 5 end-to-end links. Specifically, every two nodes in each row of the table form one end-to-end link, and indexed from 1 to 5. In each row, the former node was used as the server and the latter one was used as the client. The throughput and latency for communications with and without RPAH (i.e., static communication) were measured as comparisons. First, the throughput and latency of the 5 links without launching RPAH were measured, which reflected the basic network performance of the static communication on the selected network links. Then, the same measurements were performed again after launching RPAH, which reflected the performance of the hopping communication in the same network links. The results are shown in Tables 2 and 3, respectively. Throughput data was the average of 10 Iperf [34] sessions. Mean round trip times (RTT) were obtained by sending short TCP messages back and forth between the nodes of each flow 100 times. From the tables, the impact of introducing RPAH mechanism on latency (average overhead $\approx 4.09\%$) is much less significant than its influence on throughput (average

Table 3 Latency overhead

Static Mean RTT (ms)	RPAH Mean RTT (ms)	Overhead (%)	Max RTT (ms)	Max Overhead (%)
56.84	60.63	6.60	62.10	9.25
232.92	241.57	3.71	248.72	6.78
129.45	132.34	2.23	138.21	6.77
323.24	340.45	5.32	351.39	8.70
425.44	436.37	2.57	440.05	3.43

overhead $\approx 14.33\%$). The drop in throughput is not only due to address and port encoding/decoding and translation, but is also a result of access control and demultiplexing in RPAH.

RPAH provides an efficient hash table based port and address hopping and translation, and the implementation and test in Sect. 5 simply proved that the RPAH is a feasible port and address hopping approach. We think the scalability of our method against the number of concurrent connections can be guaranteed through efficient hash table based port and address hopping and translation as well as the acceptable bandwidth and latency overhead. In order to prove this, we use *ApacheBench* to perform 1000 web requests with concurrences 50, and perform the same throughput and latency measurements at the same time. The results are approximately consistent with the original data even though with concurrent connections more than 50. Therefore, the proposed method is able to provide hopping communications for various services simultaneously.

7. Conclusion and Future Work

In this paper, we present a novel moving target network defense framework, called Random Port and Address Hopping (RPAH), which integrates source, service and temporal hopping to effectively thwart various network reconnaissance and threat models. Our approach can achieve the maximum hopping diversities by introducing the port and address demultiplexing mechanism to extend the available ports and addresses to the whole port and address space. RPAH also provides us with a convenient attack detection mechanism with which the attack attempts using invalid or inactive ports and/or addresses will be detected and denied because of mismatching the access control list.

To effectively evaluate our technique, we performed rigorous theoretical analysis and simulation. The simulation results show that RPAH can effectively thwart scanning tools by invalidating at least 37% of their discovery in the worst-case scenario, and RPAH can thwart the scanning worm by slowing down its propagation speed by several decouple to 100 times compared to the existing works and the static network. We also conducted experiments to evaluate its effectiveness and overhead using practical implementations on both our university campus network and the PlanetLab platform, the results show that RPAH functions well while introducing an acceptable operation overhead.

In the future, we plan to further study reliability and

security issues of RPAH such as failures and attacks on RPAH components. Moreover, we will investigate the effectiveness of our approach against other attack models such as DoS/DDoS and advanced persistent attacks.

Acknowledgments

This work is funded by Research Fund for the National Key Basic Research Program of China (973 Program 2012CB315906) and NSF of China (61303264).

References

- [1] Networking and Information Technology Research and Development, "National Cyber Leap Year Summit 2009: Co-Chairs' Report (2009)," https://cyber.harvard.edu/cybersecurity/National_Cyber_Leap_Year_Summit_2009,_Co-Chairs%27_Report, accessed May 23, 2016.
- [2] P.K. Manadhata and J.M. Wing, "A formal model for a systems attack surface," *Advances in Information Security*, vol.54, pp.1–28, 2011.
- [3] P.K. Manadhata and J.M. Wing, "An attack surface metric," *IEEE Trans. Softw. Eng.*, vol.37, no.3, pp.371–386, 2011.
- [4] G.S. Kc, A.D. Keromytis, and V. Prevelakis, "Countering code-injection attacks with instruction-set randomization," *Proceedings of the ACM Conference on Computer and Communications Security (CCS'03)*, 2003.
- [5] S. Antonatos, P. Akritidis, E.P. Markatos, and K.G. Anagnostakis, "Defending against hitlist worms using network address space randomization," *Computer Networks*, vol.51, no.12, pp.3471–3490, 2007.
- [6] S. Bhatkar and R. Sekar, "Data space randomization," *Lecture Notes in Computer Science*, vol.5137, pp.1–22, 2008.
- [7] E. Al-Shaer, "Toward network configuration randomization for moving target defense," *Advances in Information Security*, vol.54, pp.153–159, 2011.
- [8] M. Crouse, E. Fulp, and D. Canas, "Improving the diversity defense of genetic algorithm-based moving target approaches," *Proceedings of the Moving Target Research Symposium*, 2012.
- [9] T. Peng, C. Leckie, and K. Ramamohanarao, "Survey of network-based defense mechanisms countering the dos and ddos problems," *ACM Computing Surveys*, vol.39, no.1, 2007.
- [10] L. Tan and T. Sherwood, "A high throughput string matching architecture for intrusion detection and prevention," *Proceedings of the 32nd International Symposium on Computer Architecture*, pp.112–122, 2005.
- [11] H. Wang, Z.H. Gong, Q. Guan, and B.S. Wang, "Detection network anomalies based on packet and flow analysis," *Proceedings of the 7th International Conference on Networking (ICN 2008)*, pp.497–502, IEEE, 2008.
- [12] H.C.J. Lee and V.L.L. Thing, "Port hopping for resilient networks," *Proceedings of the IEEE 60th Vehicular Technology Conference*, vol.5, pp.3291–3295, 2004.
- [13] G. Badishi, A. Herzberg, and I. Keidar, "Keeping denial of service attackers in the dark," *Lecture Notes in Computer Science*, vol.3724, pp.18–32, 2005.
- [14] G. Badishi, A. Herzberg, and I. Keidar, "Keeping denial of service attackers in the dark," *IEEE Transactions on Dependable and Secure Computing*, vol.4, no.3, pp.191–204, 2007.
- [15] G. Badishi, A. Herzberg, and I. Keidar, "Denial of service protection with beaver," *From Security to Dependability - Proceedings of Dagstuhl Seminar*, pp.1–6, 2007.
- [16] M. Sifalakis, S. Schmid, and D. Hutchison, "Network address hopping: a mechanism to enhance data protection for packet communications," *Proceedings of the IEEE International Conference on Communications*, pp.1518–1523, 2005.
- [17] M. Dunlop, S. Groat, W. Urbanski, R. Marchany, and J. Tront, "MT6D: A moving target ipv6 defense," *Proceedings of the 2011 Military Communications Conference (Milcom-2011)*, pp.1321–1326, 2011.
- [18] J.H. Jafarian, E. Al-Shaer, and Q. Duan, "Openflow random host mutation: Transparent moving target defense using software defined networking," *Proceedings of the HotSDN workshop at SIGCOMM'12*, pp.127–132, 2012.
- [19] E. Al-Shaer, Q. Duan, and J.H. Jafarian, "Random host mutation for moving target defense," *Proceedings of the 8th International Conference on Security and Privacy in Communication Networks*, vol.106, pp.310–327, 2013.
- [20] J.H. Jafarian, E. Al-Shaer, and Q. Duan, "Spatio-temporal address mutation for proactive cyber agility against sophisticated attackers," *Proceedings of the MTD workshop at CCS'14*, pp.69–78, 2014.
- [21] M. Atighechi, P. Pal, F. Webber, and C. Jones, "Adaptive use of network-centric mechanisms in cyber-defense," *Proceedings of 6th IEEE Int'l Symp. Object-Oriented Real-Time Distributed Computing*, pp.183–192, 2003.
- [22] L. Shi, C. Jia, S. Lü, and Z. Liu, "Port and address hopping for active cyber-defense," *Lecture Notes in Computer Science*, vol.4430, pp.295–300, 2007.
- [23] L. Shi, C. Jia, and S. Lü, "Full service hopping for proactive cyber-defense," *Proceedings of the IEEE International Conference on Networking, Networking, Sensing and Control*, pp.1337–1342, 2008.
- [24] R. Tu, J.S. Su, and F. Chen, "Network access control mechanism based on locator/identifier split," *Proceedings of the 2009 IEEE International Conference on Networking, Architecture, and Storage*, pp.171–174, IEEE, 2009.
- [25] "The netfilter.org project." <http://netfilter.org/>. Accessed Feb. 29, 2016.
- [26] "Apachebench." <http://httpd.apache.org/docs/2.0/pro-grams/ab.html>. Accessed March 18, 2016.
- [27] PLANETLAB, "An open platform for developing, deploying, and accessing planetary-scale services." <http://www.planet-lab.org/>. Accessed Oct. 17, 2015.
- [28] N. Hoque, M.H. Bhuyan, R.C. Baishya, D.K. Bhattacharyya, and J.K. Kalita, "Network attacks: Taxonomy, tools and systems," *Journal of Network and Computer Applications*, vol.40, pp.307–324, 2014.
- [29] "Nessus." <http://www.nessus.org>. Accessed March 5, 2016.
- [30] Y.-B. Luo, B.-S. Wang, and G.-L. Cai, "Analysis of port hopping for proactive cyber defense," *International Journal of Security and its Applications*, vol.9, no.2, pp.123–134, 2015.
- [31] Z. Durumeric, E. Wustrow, and J.A. Halderman, "ZMap: Fast internet-wide scanning and its security applications," *Proceedings of the 22nd USENIX Security Symposium*, 2013.
- [32] C.C. Zou, D. Towsley, and W. Gong, "On the performance of internet worm scanning strategies," *Lecture Notes in Computer Science*, vol.63, no.7, pp.700–723, 2006.
- [33] C.C. Zou, L. Gao, W. Gong, and D. Towsley, "Monitoring and early warning for internet worms," *Proceedings of the 10th ACM Conference on Computer and Communications Security (CCS'03)*, 2003.
- [34] "Iperf." <https://iperf.fr/>. Accessed Oct. 20, 2015.

Yue-Bin Luo received the B.S. degree from Tianjin University, China, in 2010, and the M.S. degree from National University of Defense Technology, China, in 2012. He currently pursues the Ph.D. degree in the College of Computer, National University of Defense Technology, Changsha, China. His research interests include intrusion detection, active defense, and network and information security.

Bao-Sheng Wang is a professor and doctoral supervisor in the College of Computer, National University of Defense Technology, where he received his B.S., M.S. and Ph.D. degrees in 1992, 1995 and 2005, respectively. His research interests include computer architecture, routing protocol and cyber security.

Xiao-Feng Wang is an assistant professor in College of Computer Science, National University of Defense Technology, where he completed a Ph.D. degree in 2009. His current research interests are in trust and security for networking systems, and distributed and intelligent data processing. He has published several papers in renowned journals and conferences such as IEEE/ACM CCGrid, AINA, IEEE Transactions on Services Computing, and Elsevier FGCS.

Bo-Feng Zhang is an assistant professor in College of Computer Science, National University of Defense Technology, where he received his B.S., M.S. and Ph.D. degrees in 2000, 2002 and 2007, respectively. His research interests include machine learning and data mining.

Wei Hu received the B.S. degree from PLA University of Science and Technology, China, in 2004, and the M.S. degree from National University of Defense Technology, China, in 2010. He currently pursues the Ph.D. degree in the College of Computer, National University of Defense Technology, Changsha, China. His research interests include high performance computing and machine learning.