PROPAGATION MODEL FOR PERMUTATION-SCANNING WORMS BASED ON DISCRETE TIME SYSTEM

[1]SAI DIVYA KALAGATLA, [2]RAMANA REDDY B., [3]MOHANA ROOPA Y.

[1]Post-Graduate Student-M.Tech., [2]Asst. Prof., Head and [3]Assoc. Prof., Dept. of Computer Science, Annamacharya Institute of Technology & Sciences, JNTUA,Tirupati-517520
E-mail: erikisaidivya@yahoo.com, bandi_ramanareddy@yahoo.co.in mohanaroopa@rediffmail.com

Abstract- In recent years, both sophistication and damage potential of Internet worms have increased tremendously. Since a new type of worm is written for each new exploit discovered, the only way to make network secure is development of systems that can react fast enough so that the spreading of any new type of worm is curtailed before it does much damage. To understand their threat, it is needed to look into their payload for signatures as well as propagation pattern for Internet-scale behavior. Traditionally, most modeling work in this area concentrates on the relatively simple random-scanning worms. Modeling the spread of permutation scanning worms can help us understand how these worms spread, and how we can monitor and defend against the propagation of worms effectively. A mathematical model is proposed that precisely characterizes the propagation patterns of the general permutation-scanning worms. In order to evaluate the effects of hit-list and permutation scanning a simulator is modeled based on discrete time system with vast internet and tested for all possible different cases. Through simulation setup we analyze that permutation scanning worms can infect the hosts in the network very fast and effectively compared to random scanning worms

Keywords- Worm attacks, Discrete Time System and Permutation-Scanning.

I. INTRODUCTION

Computer worm consists of a process or set of processes that replicates without any human intervention by creating an executing copy of itself on another computer via some form of network communication. To better understand the characteristics of DCS worms and potential countermeasures through both analysis and simulation to shown a traditional RS worms. Fast Internet worms are becoming a relatively new threat to Internet infrastructure and hosts very quickly. These worms can compromise number of hosts in a short time down to minutes now a day and these hosts can be used to perform other attacks, like massively Distributed Denial-of-Service attacks. Also the fact that there is no perfect solution to this problem as each worm is unique in its own way of exploiting a new vulnerability each time one is announced or a patch is released has made the study worm one of the challenging topics in security now a days.

Worms interest security analysts immensely due to their ability to infect millions of computers in a very short period of time. Unlike their viral predecessors, worms are network-aware and self-propagating, greatly increasing their potential reach and impact. Worms are not just a problem for those who become infected; they regularly cause problems for the uninfected as well through secondary effects such as increased network load. The propagation characteristics of a worm help to understand and possibly counter the damage potential of worms, it is very important to characterize their overall propagation properties.

A. Existing Infection Strategies

Linear scanning, where the worm scans a linear address range and partitions this range between itself and any newly infected machine is a strategy which is not seen in practice. It lacks the good initial scattering of random scanning and an intelligent firewall should immediately detect and halt linear scans. Finally, virulent worms often start by scanning the local subnet and logically adjacent networks for new victims. This is a very effective for infecting nearby machines.

B. New Infection Strategies

In a permutation scan, all worms share a common pseudo random permutation of the IP address space. Such a permutation can be efficiently generated using any block cipher of 32 bits with a pre-selected key: simply encrypt an index to get the corresponding address in the permutation, and decrypt an address to get its index. Worms infected during the hit list phase or local subnet scanning start just after their point in the permutation and scan through the permutation, looking for vulnerable machines.

Whenever it sees an already infected machine, it chooses a new, random start point and proceeds from there. Worms infected by permutation scanning would start at a random point. This has the effect of providing a semi coordinated, comprehensive scan while maintaining the benefits of random probing. Each worm looks like it is conducting a random scan, but it attempts to minimize duplication of effort. This keeps the infection rate higher and guarantees an eventual comprehensive scan. Worms pose heavy threaten to networks. Worms exploit common vulnerabilities in member hosts of a network and

spread topologically in the network, a potentially more effective strategy than random scanning for locating victims. Considering that the topology of networks has important effect on active worm spreading, it is very difficult to model propagation of active worms. For this reason, so far few propagation models are proposed which mainly concentrates on simple random-scanning worms.

II. MOTIVATION

Study of the behavior of a worm during it propagation phase is important for following reasons.

- Threat analysis with regard to speed and number of hosts a worm can compromise in a specific time frame.
- Creation of early warning system that can detect a propagating worm, with ideal case also being able to give preliminary propagation analysis and perhaps even a captured specimen.
- Prediction and analysis of collateral damage.
- Help reverse-engineer an existing worm.
- Design counter-worms or beneficial worms.
- Till now only a small portion of the possible parameter space has been used by current worms, so it is very important to know what might happen in other cases and be prepared.
- Understand effects of worms on networks.
- Determine behavior of possible future worms.

These are the points that act as a strong motivation to use simulation or some methodologies which help in better prepared for future worm attacks.

III. PERMUTATION SCANNING VARIANTS

The primary motivation behind permutation scanning is to increase the speed of an epidemic by avoiding redundant address selection among probing hosts. However, the technique also offers the potential to reduce duration and scan traffic when used in conjunction with divide-and-conquer methods. Four permutation scanning variants of self-stopping heuristics are evaluated.

A. Greedy Permutation

If the host achieves a redundant hit, it will randomly choose a new seed and continue. No hosts stop. Reseeding is motivated by the observation that another host is already scanning forward in current permutation space any continued scanning with the same seed would be a duplicate effort.

B. Stop-k Permutation

Upon achieving a redundant hit, an infecting host will deactivate with probability 1/k.

C. Sum-Count-X Permutation

Reseed-upon-redundant-hit policy used by Greedy Permutation

D. Partitioned Permutation

Each infected host is given responsibility for scanning a disjoint interval of the permutation space. Upon achieving a new hit, a host assigns half the un-scanned portion of its current interval to the newly infected descendant, and continues scanning the other half. A host stops when it has finished scanning its interval

A data flow diagram provides no information about the timing of processes, or about whether processes will operate in sequence or in parallel. It is therefore quite different from a flowchart, it allows a reader to determine not what kinds of data will be input to and output from the system, where the data will come from and go to, where the data will be stored. Worm propagation can be broadly described by process illustrated in Fig 1.

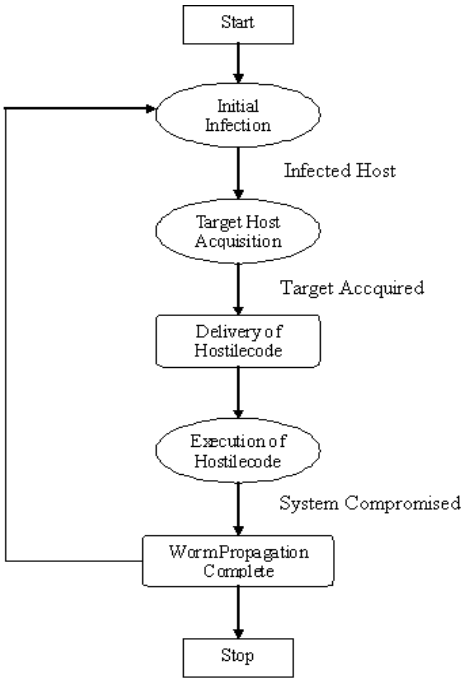


Fig. 1. Data flow diagram for Worm Propagation

In the Initial Infection the model begins with the presumption that there exists a system that is already infected by the worm and that the worm is active on this system and in Target Acquisition, in order for the worm to propagate itself it must find additional systems to infect. The presence of hostile code on a system is not sufficient for worm propagation; execution of the code must be triggered in some fashion which is done in Execution Step. This process is repeated in a loop till required numbers of hosts are compromised.

A sequence diagram in Unified Modeling Language (UML) is a kind of interaction diagram that shows how processes operate with one another and in what order. The propagation of worms between the end hosts as shown in Fig 2.

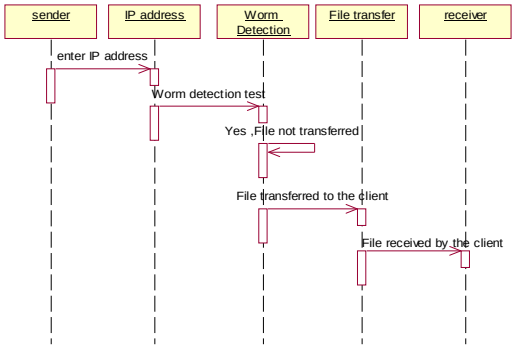


Fig. 2. Sequence diagram for Worm Propagation

A sequence diagram shows, as parallel vertical lines, different processes or objects that live simultaneously and as horizontal arrows, the messages exchanged between them, in the order in which they occur. This allows the specification of simple runtime scenarios in a graphical manner.

IV. PROPAGATION MODELING

The propagation model of a worm reflects the fractions of vulnerable hosts that are active, infected (total) and retired (inactive) worms over time. A scan message that does not hit any vulnerable host does not change these numbers. Thus, modeling should only be based on the event of a scan message hitting a vulnerable host. When that event happens, all aforesaid numbers change. The simulator is implemented with proper encapsulation, that is a host object inside the simulator is not aware of the large picture of the network, and instead it can only see its own private variables, including its IP address, the state of its local random-number generator, the last address scanned, and the response to a scan message, that is new infection or not.

A. Network Module

Simulator model fully-connected networks, using network in a more graph theoretic sense rather than a TCP/IP sense. Network models all the wiring and routers and gateways that compose a single real network. Every network-connected entity can send a message to any other entity in the simulation, each network-connected entity has an address consisting of a single number and the user has to set the address space size. The instance of Network contains and connects instances of class Host.

B. Host Module

Each host object performs the following operations: If it is not infected, it does nothing. If it is infected, it uses a random number generator to produce a random location on the permutation ring and begins to sequentially scan the addresses along the ring after that location at a certain rate. Each object of this type has a numerical address associated with it and can contain zero or more software instances. It can contain actual code that gets executed. Simulator will add an instance of software to the host and give software code a chance to initialize itself and also remove a software instance from that Host. The valid address does not equate to this Host's address either instead it have to pass this up to the Network object this host belongs since network object constitutes the only thing in a simulation that knows size of address space.

C. Messages Module

It consist Sending and Receiving Message objects and sees if it matches the "architecture" of the current Host and then see if the current Host has the destination Software in it.

A Message gets delivered to a host by numerical address, but the Message gets ignored if the Message instance does not contain an identifier that matches the identifier of the Host instance receiving it. After a match on Host identifier, the Host will ignore the Message instance if the Message does not name a Software instance the Host contains. Instances of class Message model frames or packets, and constitute the hunks of information that Host objects can pass between each other.

D. Permutation Module

Permutation scanning worms don't select addresses to probe randomly, hosts select addresses algorithmically. The permutation scanning worm of this simulation uses an algorithmic permutation to probe address space. A DES-inspired algorithm combines a 32-bit key with a 16-bit index to determine the address to probe. Worms start with a random index, and then generate an address to probe by using the permutation function on the index. Worms increase the index sequentially and calculate different, non-sequential addresses to probe. The series of addresses calculated by permuting sequential indexes look random. All worms use the same permutation by using the same 32-bit key and the same algorithm to generate addresses, but individual worms pick random initial index values at infection time. The 4-byte key actually gets used as 4, single-byte sub-keys. There are 4 rounds of combining input bytes with key bytes. This is very similar to DES but the difference coming from what function f does to key and the right-hand half of the value.

E. Quantities using Probabilistic Approach

When an effective host hits a vulnerable host, let denote the probability for the vulnerable host to be a new (old) infection. We observe that an effective host can hit only two types of vulnerable hosts: Those that are uninfected and infected ones that are the tails of scan zones. To avoid re-probing, worms that try to re-infect previously infected hosts go inactive. The

worm code in the permutation scanning simulation accomplishes this by replying to any re-infection attempts. Individual re-infecting worm instances go inactive when they receive a reply.

This module have to distinguish between another Permutation scanning worm trying to re-infect the Host, and an already infected Host answering a probe from this Host and a previously infected Host just probed this Host and Send a Message back to indicate infection. When the host just probes a previously infected Host then reset sequence number, and goes inactive if this is the Nth reply received. You only need to count replies from previously infected Hosts if you want to keep probing after getting a reply

V. RESULTS

The simulation is performed using Eclipse and Activeperl tools. The simulation runs in an address space of size 65535 (16 bits). It starts with 7 permutation scanning worm and 9993 victims. The victim Host instances reside at randomly assigned addresses, as does the single initial worm instance. The simulation runs for 250 time steps with a difference of 5 steps between two steps. Simulator assumes complete connectivity within address space using a pseudo-random permutation to map addresses to a subset of vulnerable machines. Worm propagation happens among end-hosts. The permutation scanning worm simulation is run several times and calculated mean counts of infected hosts at every fifth time step, shown in the graph below in Fig 3. The simulation results are tabulated as shown in Table 1-3.

Table. 1. Simulation results-1

Step	Msg Count	Base Victims	Active Worms	Inactive Worms	Total Worms
5	41	9997	11	1	12
10	76	9987	21	1	22
15	161	9965	43	1	44
20	294	9935	73	1	74
25	510	9876	131	2	133
30	957	9761	242	6	248
35	1899	9506	492	11	503

Table. 2. Simulation results-2

Step	Msg Count	Base Victims	Active Worms	Inactive Worms	Total Worms
5	41	9997	11	1	12
10	71	9987	21	1	22
15	128	9975	33	1	34
20	277	9935	73	1	74
25	520	9880	128	1	129
30	979	9750	256	3	259
35	1910	9498	502	9	511

Table. 3. Simulation results-3

Step	Msg Count	Base Victims	Active Worms	Inactive Worms	Total Worms
5	40	9997	11	1	12
10	73	9989	19	1	20
15	149	9966	42	1	43
20	341	9915	93	1	94
25	679	9836	172	1	173
30	1265	9682	326	1	327
35	2472	9375	620	14	634

The simulation curves are plotted using the averaged data collected from the simulator. The model curves in the Fig 3., show the percentages of vulnerable hosts that are active, inactive and total over time, respectively. The behavior of permutation scanning worm. It gets a huge boost from the hit-list during the first few seconds of propagation, quickly establishing itself on the network and then spreading exponentially. As the infection exceeds the half point, some of the worms begin recognizing that saturation is occurring and stop scanning. By the time the graph ends, most of the worms have gone silent, leaving a few remaining worms to finish scanning the last of the address space.

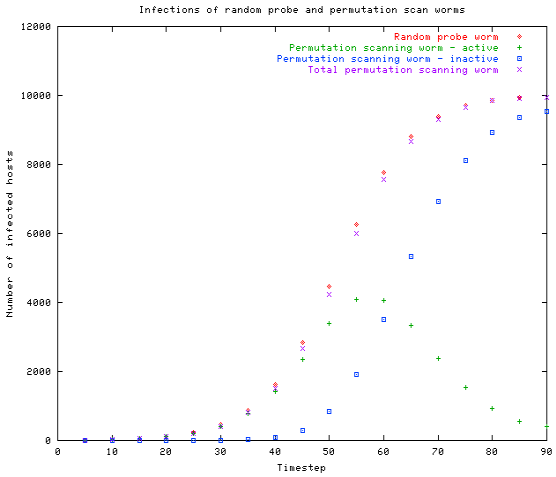


Fig. 3. Infections of random probe and Permutation scan worms

Permutation scanning worms and random probing worms take exactly the same number of time steps to reach full infection but Permutation scanning produces fewer messages to get to full infection than a population of random probing worms does, and a smaller mean number of probes per host. Only active permutation scanning worms send messages, and the proportion of active permutation scanning worms rises, and then drops to nearly zero by full infection. To contrast, every member a population of random probing worms produces messages all through the course of a simulation.

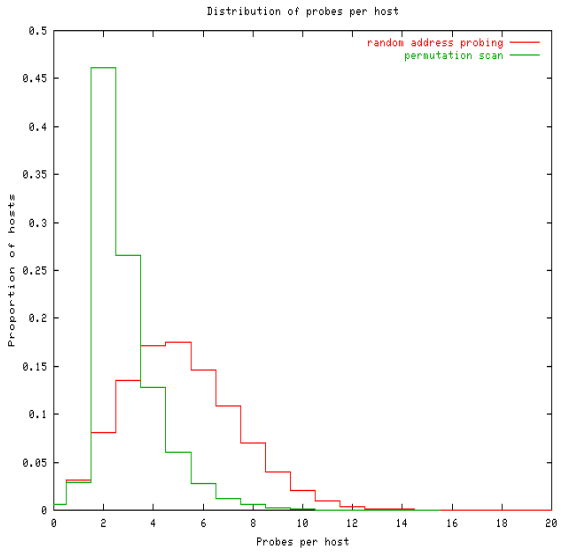


Fig. 4. Distribution of probes per host

The Fig 4. shows distribution of probes per host, one area of difference exists between permutation scanning worms and random probing worms: the distribution of probes over the population of entities in the simulation. The number of hosts receiving a certain number of probes amounts to almost exactly half the number of hosts that receive one more probe. The typical host receives far fewer scans for the optimistic permutation scanning worm, than for a random scanning worm. The property of not probing most hosts very often might make discerning a permutation scanning worm more difficult, but gives advantage of being stealth.

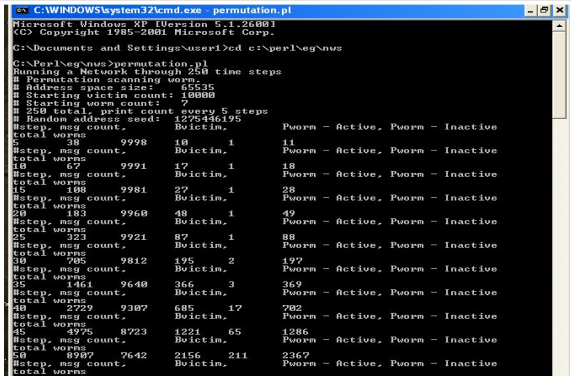
CONCLUSION

In this paper, we have successfully modeled the propagation characteristics of permutation-scanning worms. We first derive the precise propagation model for permutation scanning worms. To verify the correctness of the model, we compare the results from our model with those obtained from actual worm simulations and show that they perfectly match. It is concluded that permutation scanning does give a worm an overall advantage relative to random probing.

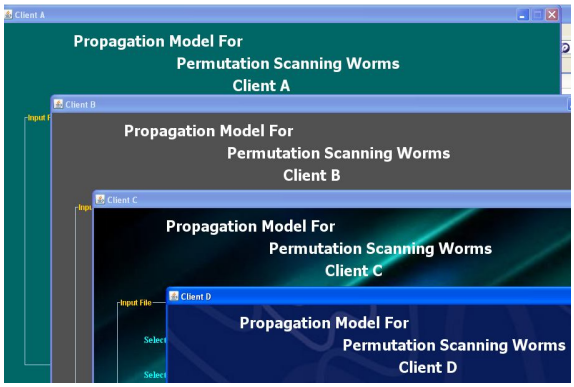
Through simulation setup we analyze that permutation scanning worms can infect the hosts in the network very fast and effectively compared to simple random scanning worms.

Outputs:

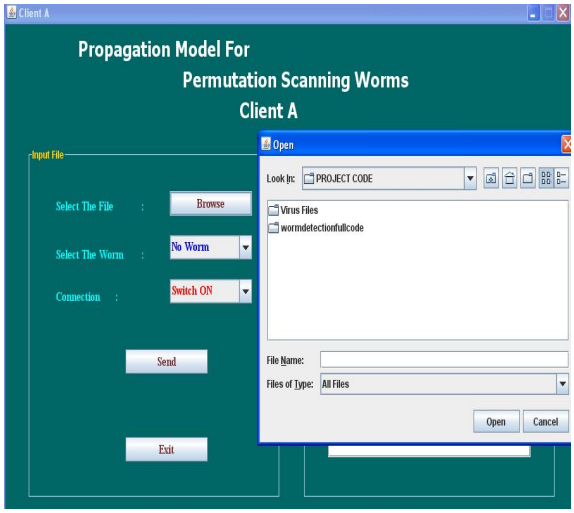
The Fig B.1 shows the output obtained after running source code in dos window. The output obtained shows the varying values of worm at different steps. It also shows different other parameter values which are used in the simulation.



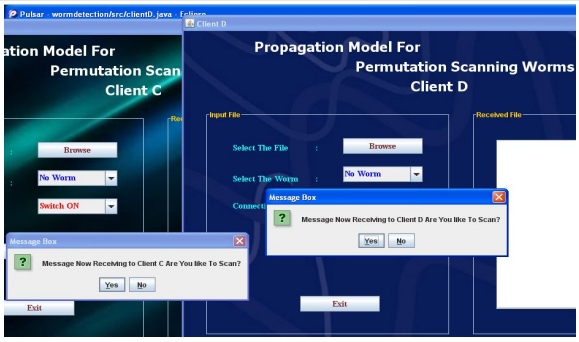
The Fig B.1 shows the output obtained after running source code in dos window. The output obtained shows the varying values of worm at different steps. It also shows different other parameter values which are used in the simulation.



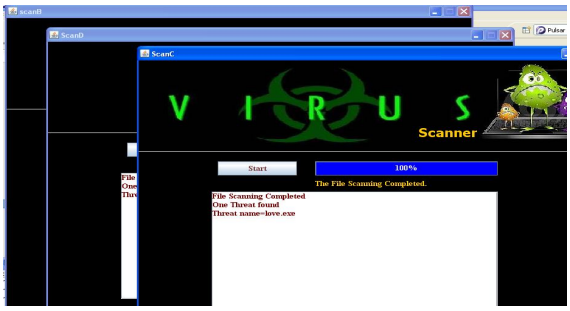
The code is run on Eclipse tool, all the hosts namely A, B, C, D are opened, each client contains buttons which are used to start propagation and each host is switched on or off based on the requirement.



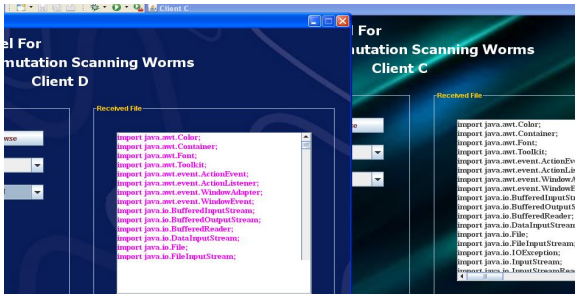
After switching on the clients any one of the host is selected in which some data files are browsed using the browse button and a worm is attached to the file using select worm and then clicking on the send button sends the infected file to other hosts.



The host scans the other hosts connected to the network to send the file, and other hosts ask whether to accept it by popping a message box, to accept to receive file click on yes else no.



When the files are accepted to be received the built in virus scanner inside the hosts pops up. Click start to start the scanning for virus, after scanning gets completed the results are shown on the text window by which worm propagation found. Remove button is used to remove worm and Exit button to exit from scanner window.



Files are received without the worm and are displayed at the received file window screen of the hosts.

REFERENCES

[1] C. Chen, Z. Chen, Li Yubin, "Characterizing and defending against divide-conquer-scanning worms," Computer Networks, vol. 54 pp. 3210-3222, 2010.

[2] Y. Li, Z. Chen, and C. Chen, "Understanding divide-conquer-scanning worms": Proceeding of the 27th IEEE International Performance Computing and Communication Conference (IPCC'08), Austin, TX, 2008.

[3] Z. Chen, C. Chen, and C. Ji, "Understanding localized-scanning worms": Proceeding of the 26th IEEE International Performance Computing and Communication Conference (IPCC'07), pp. 186-193, New Orleans, LA, 2008.

★★★