Agent-Based Host Enumeration and Vulnerability Scanning Using Dynamic Topology Information

Ziyad S. Al-Salloum
Information Security Group
Royal Holloway, University of London
Egham, Surrey TW20 0EX, UK
Email: z.al-salloum@rhul.ac.uk
and
Prince Muqrin Chair of IT Security (PMC),
King Saud University, P.O. Box 2454,
Riyadh, Saudi Arabia

Stephen D. Wolthusen
Norwegian Information Security Laboratory
Gjøvik University College
N-2818 Gjøvik, Norway
and
Information Security Group
Royal Holloway, University of London
Egham, Surrey TW20 0EX, UK
Email: stephen.wolthusen@rhul.ac.uk

***Abstract*—Edge networks in enterprise networks are increasingly complex and dynamic, raising questions about the ability to maintain a current overview of computing assets on the network and their potential vulnerability. However, to respond to ongoing or impending attacks that may propagate at high speed, it has become crucial to ensure proper and efcient reachability of all network nodes that might be at risk so as to be able to assess and, where possible, mitigate the threat.**

In this paper we therefore propose an agent-based semi-autonomous scanning mechanism which utilizes topology information to traverse networks with minimum bandwidth usage and maximum network coverage, and hence avoiding potential service degradation in large-scale structured networks. Topology information is also used to constrain propagation to a well defined network, while intermittently active hosts and topology changes are detected by using resident reactive agents plotted throughout the mechanism gradual propagation.

I. Introduction

The constant threat of service disruption due to network attacks has been a major concern to many organizations. Especially, with these attacks increasing [1], while most of them have exploited an already publicly known vulnerability before it has been addressed by network administrators. For example, in case of Conficker, the relevant security update has been published in October of 2008 whilst different Conficker versions observed in November 2008 through April 2009 have exploited the same vulnerability [2], [3]. The delay in distributing updates might be explained by the quality of security-related configurations, but the question of how efficient current mitigation mechanisms in providing sound protection to enterprise networks does still remain. Indeed, in many cases it is crucial to respond promptly and effectively to eliminate a malicious worm outbreak. Self-replicating agents are one way to efficiently reach network nodes, as their self-discovering nature enables them to find and detect even undocumented and hidden nodes [4], especially common under dynamic and complex networks. However, besides the possibility of self-replicating programs being vulnerable to misuse [5], such an approach can consume significant bandwidth resulting in unacceptable service degradation as well as hindering the discovery and mitigation mechanism itself. To address this issue, we proposed an agent-based mechanism which uses topology information found in structured networks to optimize its propagation speed whilst minimizing resource consumption [6]. And in this paper we introduce enhancements and improvements to address the limitations of our mechanism which include: Edge node failure recovery, network backbone traverse, and intermittent node detection and recovery. The remainder of this paper is structured as follows: Section II briefly reviews related work on different worms propagations, while section III introduces topology and modeling assumptions. Section IV compares self-replicating with classical based vulnerability detection, after which Section V introduces improvements of the mechanism, followed by a discussion on treating intermittently nodes and topology changes in section VI, whilst mechanism design aspects are then described at section VII. A short overview of vulnerability mitigation techniques is described in section VIII followed by risks and threats to the mechanism in section IX, then a comparison of a naïve scanning strategy in section X, whose properties are evaluated through network simulation in section XI before a brief discussion of simulation results in section XII. Finally, our conclusions are then described in section XIII.

II. Related Work

Since their creation computer worms have adapted different techniques to allocate vulnerabilities and propagate to cover as many targets as possible. Worms like Code Red I and Slammer have used random scanning techniques to allocate their targets, while even the Morris worm used a more intelligent way of propagation [7]. The propagation speed increases when the worm incorporates information of all the vulnerable hosts such as in Flash worms [8]. A hit-list worm would incorporate information about some vulnerable hosts where then it will switch to random scanning after scanning all the hosts in the hit-list as proposed by Staniford *et al.* with further targeting enhancements suggested e.g. by Fan and Xiang [9]. Worms like SQLsnake uses a built-in list of numbers that will be used later to generate network addresses to probe for vulnerabilities, these numbers are generated according to network space that most probably contain vulnerable targets [10]. Zou *et al.* proposed a *routing worm* which scans based on the information provided by Border Gateway Protocol to reduce scanning space [11]. They also introduced a *divide-and-conquer scan worm* which uses the divide and conquer approach to propagate; when the target is infected it passes half of the scanning space to the target and continues scanning the other half of its original space [12]. Code Red II used different scanning techniques where hosts closer to the infected target are scanned with higher probability than those farther away [13]. This is a scanning technique referred to as *Island Hopping*, as the network is looked at as islands where an island receives specific attention before hopping to another island [10]. Vojnović *et al.* identified optimal static and dynamic propagating strategies. These proposed strategies minimize the total number of sampling to reach a target fraction [14]. In one strategy, a worm infects a randomly selected host then tries to spread on the same subnet, as long as there are many vulnerable hosts. If not, the worm shift to another subnet using random scanning [15]. Markus Kern has introduced CRclean, a worm that spread passively by listening to Code Red I scanning attempts. When Code Red I scan a host that already has CRclean installed, CRclean will responds to the scanning activity by reinfecting the scanning source host and removing the malicious worm to provide remediation and containment of Code Red I malicious spread [16]. The worm, however, has not been tested in real world networks. Blacklists were used by Conficker, the list contained entities that might provide remedies and containment actions towards malicious code, such as anti-virus sites and Microsoft, the worm will not try to scan IPs in the list to avoid detection [17]. Al-Salloum *et al.* proposed an agent-based (vulnerability) scanning mechanism using semi-autonomous propagation strategies similar to those found in worms, but which utilizes information from the link layer (layer 2 of the OSI model) to reconstruct topology information found through the *Link Layer Discovery Protocol* to detect neighboring nodes and propagate gradually until total coverage of an enterprise network is reached [18]. They also introduced a Link-Layer-Based Self-Replicating Vulnerability Discovery mechanism which utilizes topology information such as Content-addressable memory (CAM) tables and Spanning Tree information stored in active network components such as switches. In this approach, the scanning mechanisms propagates through traversing active network components whilst probing vulnerable hosts until the network is covered [6].

III. Network Topology Model

In the design of the network we have taken into account where the mechanism will most probably and most beneficially be found. These networks were designed hierarchically with different number of local area networks (LANs) connected with each other through a backbone. This topology mimics to a large extent enterprise networks to both accelerate and contain scanning activity whilst making use of the disparate bandwidth available on backbone and cross-link networks as well as the considerably higher bandwidth of edge switches. For topology change and intermittently active hosts detection, each node run under two modes: online and offline. When the node is on the offline mode it can not be seen by the mechanism and is considered disconnected until it becomes online. Fig. 1 shows an example of a small 100-node network, with a switch and hosts in the offline mode. For more accurate results, different numbers and layouts of hierarchical networks have been designed for simulations. Network nodes are chosen randomly to be linked to a randomly chosen switch under a specified probability. A switch is then linked to the router in the backbone. In the simulation we have assumed the following: Simple Network Management Protocol (SNMP) is supported, network implements the Spanning Tree Protocol STP and is loop-free, network backbone implements Open Shortest Path First (OSPF) as a routing protocol, and CAM/port status is maintained by switches. These assumptions are not uncommon in enterprise networks. We have generated 17 hierarchical networks with connected nodes via a duplex-link where packets can flow in both directions, and for the results reported in section XI, the number of nodes was chosen between 100 and 8000, with link bandwidth set to 100 MBps and assuming a discovery packet size of 900 bytes. If we consider a switch domain as a switch with its directly connected hosts, then there exist three mechanism failures that can be described as: self-replicating failure to the next switch domain, failure when probing an edge node, and self-replicating failure to the next LAN. The cause of failures, however, is either due to a link or a node operation failure. In both cases the agent will try to recover the failure. More simulation attributes are mentioned in section XI.

IV. Why Self-Replicating Approach

The self-replicating approach outperforms traditional vulnerability scanners at least in three main aspects: Probing

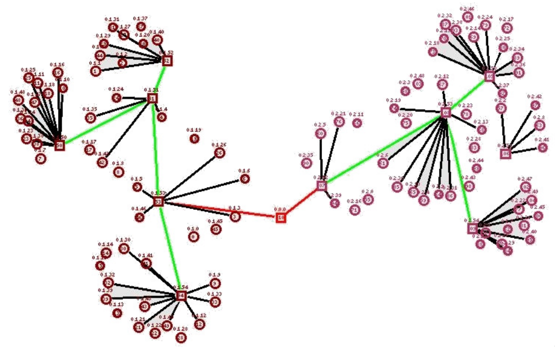

Fig. 1. A 100 node hierarchical network that implements STP with scattered disconnected (offline) nodes, including a switch.

distance, ability to detect intermittent nodes and traversing the network without regard to network architecture. A shorter communication distance between a vulnerable node and the probing server is faster and less exposed to link failures compared to communicating to a remote IP address. Self-Replicating approaches install agents along their propagation path which hire each node to participate in the scanning activity to ensure high scanning coverage. Scanners, however, scan different IP addresses remotely adding more distance and more time for the scanners to communicate with their targets. Furthermore, Self-Replicating approaches keep probing their targets for vulnerabilities and detect any newly joining nodes or off-line nodes that become on-line. Yet, when vulnerability scanners conclude their assessment, newly joining or previously offline nodes become exposed. Also, the self-discovering nature of the self-replicating approach gives it the ability to spread around complex and large network without high regard to the network architecture. Yet, scanners are usually required to be installed at each network segment for better performance [4].

V. Agent-Based Host Enumeration and Vulnerability Scanning Using Dynamic Topology Information

The Link-Layer-Based Self-Replicating Vulnerability Discovery Agent discussed in [6] is a mechanism that utilizes network topology knowledge to increase propagation efficiency by using STP and Content Addressable Memory (CAM) table to both explore topology and identify edge node status. However, the mechanism falls short in performing edge node failure recovery and topology change detection, which is detrimental when confronted with volatile edge networks such as wireless networks. It also does not provide an algorithm for traversing the network backbone, which requires assigning a host at each LAN as a starting point. Therefore, in the following we provide enhancements to address these limitations. In order to provide edge node failure recovery, the scanning mechanism will wait for a response from edge nodes to ensure successful probes; upon failure to respond, the agent will retry after 0.5 seconds.

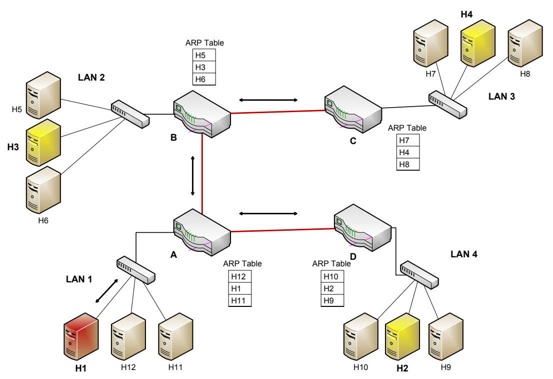

Fig. 2. Traverse Backbone Algorithm

For the scanning agent to be able to propagate from LAN to another LAN, it needs to map the topology of the backbone. Since OSPF routing protocol is implemented, the agent upon detection of a router during its propagation, sends an SNMP MIB request to fetch the OSPF Link State Database (LSD), which gives a complete description of the network, including: Routers, network segments, and how they are interconnected [19]. This database is built by the collection of Link State Advertisements (LSA) sent by each router in the backbone to describe its local routing information. The agent also reads the ARP cache of the LAN interface of each router to detect a host IP address to self-replicate to. For example, Fig. 2 shows a sample of a network of four LANs connected by a backbone that consists of four routers. When the agent reaches $H1$ and detects a router device (e.g. by checking the SNMP MIB value of *ipForwarding* if set to one then the device is a router [20]), it will send an SNMP MIB request *ospfLsdbTable* to fetch the LSD [19] and the ARP cache stored at router A using SNMP MIB *ipNetToMediaTable*; upon receiving router's A response the agent at $H1$ extracts the backbone network map, which in this example consists of routers A,B,C, and D. The agent thereafter sends an SNMP MIB request to routers B, C, and D to fetch the stored ARP cache of network interfaces connected to the LAN. In parallel, the agent picks a host in the ARP cache table of router A to detect a valid IP address that belongs to the LAN where the agent should propagate (No IP will be picked in the case of the example, since there is only one LAN connected to router A). The agent subsequently attempts to self-replicate to the IP address chosen and waits to receive replies from routers B, C, and D containing ARP cache tables. Upon receiving the tables, the agent picks one IP address for each LAN and self-replicates to it, to ensure proper coverage of the vulnerability discovery mechanism (retrying in case of replication failure, e.g. if the platform is not supported[1]). We denote the list of LANs connected to the first router encountered by the mechanism as $L(ARP_{R_0})$ where ARP_{R_0} is the ARP cache stored at router R_0. And we denote LSD stored in the first router as LSD_{R_0}, where the number of routers according to the database is $LSD_{R_0}\{N\}$. We denote the function of fetching data through SNMP as *SNMP* and agent self-replicating as *SR*. The algorithm for traversing the backbone, then follows:

1) Fetch Link State Database (LSD) and ARP cache from the first encountered router, using SNMP: $SNMP\{LSD_{R_0} \cup ARP_{R_0}\}$
2) Pick an IP address from each LAN interface (else the source LAN interface) in the ARP cache of the first encountered router and self-replicate to that IP address: $SR(L(ARP_{R_0}) - \{L_0\})$
3) Fetch ARP cache from all other routers in the backbone (according to LSD), using SNMP: $SNMP\left\{ARP_{R_1} \cup ARP_{R_2} \cup ..ARP_{R_{LSD_{R_0}\{N\}}}\right\}$

[1]The replication mechanism itself is beyond the scope of this paper; however, both managed service interfaces and active exploitation of vulnerability may be considered.

4) Pick an IP address from each LAN interface in the ARP caches of backbone routers and self-replicate to these IP addresses: $SR(L(ARP_{R_1}) \cup L(ARP_{R_2}).. \cup L(ARP_{R_{LSD_{R_0}\{N\}}}))$

The algorithm assumes that the Autonomous System consists of a single area only, where a single read of the LSD is enough to determine the map of the backbone. Dividing the backbone into more than one area is used in very large networks to reduce the size of routing tables, but can be handled in a straightforward manner.

VI. Intermittently active hosts and topology changes

In dynamic networks, it is difficult to achieve efficient coverage of a network with transient nodes without excessive scanning frequency. However, by utilizing topology information it is possible for the mechanism to determine topology changes and intermittently active hosts for vulnerability discovery. The agent takes advantage of the CAM table stored in the switch and stores it during the first scan round. After 0.5 simulation seconds, the switch is probed for the CAM table and checked for changes; new nodes or switches are thereafter detected by the agent and probed. When a new switch is listed in the CAM table the agent has to verify that it is actually a directly connected switch, by using *Lemma 4.1 and 4.2* [6]. Intermittently active hosts and topology changes detection can, therefore, be achieved by the following algorithm:

1) Fetch CAM table from the directly connected switch.
2) Compare the CAM table with the locally stored previous CAM table.
3) Probe newly detected edge nodes.
4) Probe newly detected switches devices according to *Lemma 4.1 and 4.2* [6]
5) Go back to step 1 each time *t*

Note that to speed up the simulation, we have chosen 0.5 simulation seconds for the agent to check the CAM table. However, the interval may be up to 5 minutes (the default expiration time for CAM tables stored in switches) [21]. But what about systems missing from the CAM table due to inactivity or table age expiration, those systems will be treated as offline nodes and will be detected when they become active as per the algorithm. The agent can also check for topology changes by listening to BPDUs emitted from the root switch. When a topology change occurs, i.e. when a switch port goes into forwarding status or from forwarding (or learning) into blocking, the switch will emit a *Topology Change Notification (TCN) BPDU* to the next switch (towards the root bridge) until the root switch receives the *TCN BPDU* and emits a configuration BPDU with the topology change bit on. The BPDU emitted by the root is sent to all switches and thereafter the agent can listen to it, where it can trigger a topology change detection. However, in case the agent might miss a topology change BPDU we have set the agent to check for network topology changes each 0.5 simulation second.

VII. Vulnerability Discovery Mechanism Design Components

Nazario *et al.* defined different components that constitute a worm system [22]. We use their components to describe the design of our mechanism with slight modifications. The scanning and vulnerability discovery mechanism consist of three components that allow it to cover the network as follows:

- *Reconnaissance.* This component is responsible for discovering host nodes that are vulnerable. The improved scanning mechanism reads CAM tables stored in switches to detect edge node hosts to probe for vulnerability. The mechanism also reads STP in addition to CAM table to determine next switch to propagate [6].
- *Probe Component.* This component describe the method the mechanism use to detect the property (e.g. vulnerability) at a target node. For the sake of simplicity, our simulation assumed all hosts to be susceptible and it requires only one packet of the size 900 bytes to exploit another node. However, in a heterogeneous network, more elaborate scanning will be required from agents.
- *Communication.* This component describes the communication between agents. The scanning mechanism enables communication between agents by sending an acknowledgment packet to the sender agent to only ensure that the self-replicating task has been accomplished successfully so as to disable any blocking attempts.

These three components summarize the design of the scanning mechanism; further extensions are discussed briefly in section XIII.

VIII. Vulnerability Detection

In our simulations, we have assumed that one packet is enough to detect a vulnerability and install an agent; which mimics – to some extent – SQL Slammer which uses a single packet to propagate [23]. Of course, one packet is not enough to contain a payload that performs complex tasks (or deal with multiple vulnerabilities), however, for such tasks larger payloads can be used. Different approaches do exist to detect a vulnerability such as by exploiting it. The agent will try to probe a potential vulnerable machine by sending a packet with the necessary payload to achieve three tasks. First, exploit the vulnerability to gain the necessary privilege to apply temporally remediation. Second, apply vulnerability remediation to eliminate the security exposure of the vulnerable machine. Third, trigger the agent for further propagation to cover other vulnerable nodes. Vulnerability remediation is temporary and clearly is not substitute for a code-level patch and can rang of different techniques, such as disabling a port that can be used by a malicious user to compromise a machine, or installing a wrapper script that will act as a packet filter between the vulnerable application and the network, or even uninstalling the vulnerable application. What ever remediation is used by the network security team, it must have the required privilege to assure successful deployment. The mechanism, however, should be deployed carefully as payload might be exposed to network users with malicious intention; but the mechanism will most likely be triggered by the enterprise security team in response to a critical vulnerability with an already publicly available exploit. If exploitation was successful, the node is vulnerable, otherwise, the node most likely is not. This approach would eliminate the amount of false positives, usually reported by vulnerability discovery applications [24]. However, the exploitation procedure must be performed carefully as it can lead to service disruption, where it might get the system into a halt or unstable state, bound to the nature of the vulnerability and its exploit. Tests should be conducted before deploying any exploit. However, when this cannot be ensured a secondary propagation mechanism must be used.

IX. Risks and Threats

Since the propagation path depends on topology information, any malicious interactions with STP, OSPF, ARP, SNMP, or CAM tables might have negative impact on the mechanism behavior. For example, the lack of authentication of the STP protocol, makes it possible for a malicious user to manipulate the topology and compromise the integrity of BPDUs leading to undesirable actions such as changing switch port status or electing a compromised switch as root. Yet, although these vulnerabilities does not relate to the agent directly, any countermeasure to prevent such abuse would participate in the protection of the mechanism itself. Some protection measures do exist for STP, such as disabling user ports upon detection of STP traffic and disabling ports that emit false BPDUs that elect false roots. A malicious user flooding CAM table with fake MAC addresses, will cause the switch to act as a hub, with no edge information for the agent to utilize. One possible mitigation to such misuse would be shutting down a port if more than one MAC were detected, performance issues however, should be considered when applying such mitigation. For the agent to traverse the backbone it has to read the ARP cache stored in the router. ARP cache poisoning may redirect the agent to an IP address (out of the mechanism scope) inserted by malicious users, however, the agent can be designed to ignore any IP address that does not adhere to certain attributes put by the security team. ARP cache poisoning can be achieved by sending malformed Gratuitous ARPs to the target machine, however, private VLANs can – to some extent – eliminate such abuse, as they don't allow nodes on different ports to communicate at layer two but still allow them to share the same network space. Since the agent use SNMP in its communication with network devices (e.g. switches and routers) a malicious user can intercept this line of communication and alter it according to its attack preference. When the agent, for example, probe a neighbor switch to determine if it is directly connected to the current switch, the attacker can respond with an SNMP message that indicate the switch to be a non-neighbor switch, which will cause the agent to ignore the switch, and might cause the agent itself to stop any further propagation. Another scenario, would be an attacker altering the router response to fetch the OSPF Link State Database SNMP request, giving the attacker the freedom to define the backbone map according to his intrusion preference and using it as an input to the mechanism. Even if we assumed the agents to not be vulnerable to STP, ARP, or SNMP attacks; a switch domain can be isolated from the vulnerability discovery process by turning the agent off physically (weather the intention malicious or due to human error). Also, the agent can be controlled remotely if it was running under a vulnerable operating system where an attacker can exploit or if the intruder was able to compromise a privileged user account on the agent host. Different protective measures and performance aspects of the mechanism are addressed in [25].

X. Randomly Scanning Agent

We have designed a simple random scanning worm to compare it with our mechanism in terms of bandwidth. It mimics Slammer worm that uses a random scanning technique where it targets a randomly chosen IP, by sending one UDP packet to its victims [23] but with slight difference as it operates in corporate networks specifically at layer two of the OSI model. When the worm is initiated, it propagates to all LANs in the corporate network according to a hard coded destination host at each LAN. The randomly scanning worm thereafter, will utilize the information in the host to indicate the address range the worm can randomly choose from. This is done by reading the IP address and the subnet mask, of the host. We have assumed for the simulation class A network with subnet mask 255.255.192.0. The address space thereafter consists of 16384 IP addresses; ignoring the network address and the broadcast address yields 16382 possible target hosts. The worm uses one packet of size 900 bytes to infect, and we assume all hosts are vulnerable. However, there are switches and routers that the worm might scan, with no infection as the vulnerability exist only in host nodes.

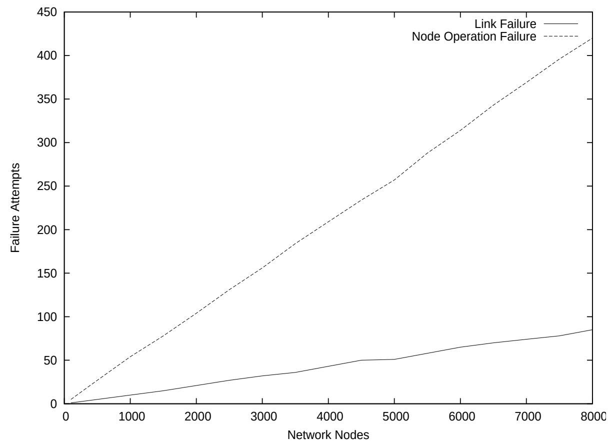

Fig. 3. Failure due to link or node operation.

XI. Simulation Results

Since each network has different topology and parameter choices, it is somewhat difficult to find a closed-form complexity estimate for the vulnerability discovery mechanism; we have therefor chosen to conduct extensive network simulations of our mechanism. For the results to be more realistic following the model and assumptions outlined in section III, hierarchical networks were used, reflecting typical network topologies in large-scale, enterprise networks. All simulations have been conducted using the Network Simulator 2 (NS-2), a discrete event simulator mainly used for research activities [26]. The simulations in total ranged to 765 hierarchical networks. Since the capacity of NS-2 is limited, we weren't able to conduct simulations on topologies over 8000 network nodes. Therefore, we have run our simulations against 100 to 8000 nodes with around 500 nodes difference between each topology. The topology does not consist of host nodes only, switches and routers also exist to reflect a multilevel topology. For more accurate results each topology has been simulated 45 times where the average has been calculated to account for random effects such as link and node operation failure in addition to time coverage. In addition to our mechanism simulation, we have performed simulations of a random scanning worm following the model and assumptions outlined in section X. However, due to the extensive amount of communications and bandwidth consumption of such random propagation strategy, the time of simulation was beyond our capacity, and therefore we settled down with one simulation of network topologies ranging from 100 to 3000 nodes.

In our simulations we have gathered the following:

- Number of link failures under $p = 0.01$.
- Number of node operations failures under $p = 0.05$. Note that node operation failures are failures caused by the node itself (e.g. system is busy or in different state due to restarting).
- Number of packets generated by both our scanning mechanism and the randomly scanning worm to cover the network. The packets, however, that are generated between a switch and a host are exempted due to it having no significant impact on the bandwidth of the network.
- Time it takes the mechanism to cover the corporate network. When there is a link or a node operation failure, the mechanism will retry after 0.5 simulation seconds.
- Time it took the mechanism to detect all new network nodes that have just joined the network. In all simulations the network nodes were randomly added to the topology at different locations. All offline nodes became online at the 2.5 second of the simulation time.
- Number of random worm scans of non-existent IP addresses.

For the sake of simplicity, we have assumed in the simulation, that the CAM table and STP information can be requested by sending one packet to the switch. One SNMP MIB request is also assumed to fetch OSPF information in addition to ARP cache table stored in the router.

XII. Discussion

It is not usual to propose a self-replicating approach as a security measure, that is due to the nature of self-replicating software; which genereally propagate randomly, consuming huge bandwidth and causing the network to become congested, as with malicious worms [27]. However, that doesn't prevent researching this area to eliminate its risks and tweak it in a way to fulfill a certain task. Our mechanism tries to combine both the distinctive exploring nature of self-replication programs and the constraints of the enterprise network, to provide a sound protection without disturbance to the network focal interests. It appears that by using the network topology information it is possible to dramatically reduce the bandwidth utilized by self-replicating programs as in the case with our proposed scanning mechanism. The mechanism before improvements [6] had some limitations in its capability to spread around the network and achieve sound coverage without human intervention (i.e. hard coding a single host IP at each LAN to bypass the backbone). The improvements thereafter

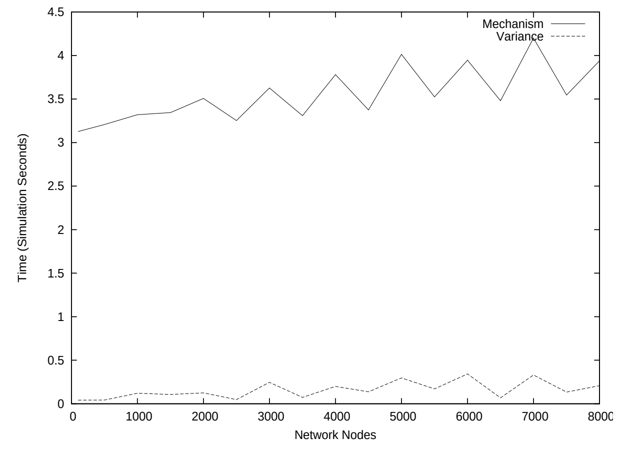

Fig. 4. Time to cover the whole enterprise network

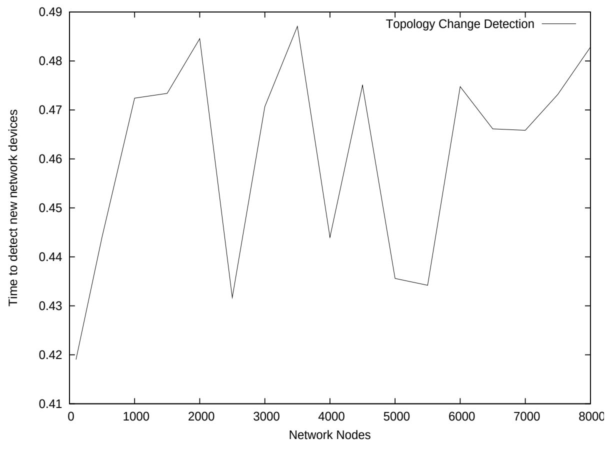

Fig. 5. Time to detect all newly added network devices

are required to address these limitations to produce a more robust vulnerability discovery mechanism. Three limitations of the previous approach have been addressed. First, it adds edge node failure recovery ensuring that all network nodes are not missed; this is important since one vulnerable node cause a threat to the whole enterprise network. Second, it provides an algorithm to detect newly added network devices, which ensures that a vulnerable node joining the enterprise network get probed as soon as possible to minimize the time of vulnerability exposure. Third, it provides an algorithm to bypass the network backbone utilizing protocols such as OSPF and ARP; which gives the mechanism the capability to spread and cover the vulnerable network as fast as possible to achieve absolute protection. Yet, as in any mechanism it is not fully immune to threats and risks. Our mechanism depends on topology information to propagate, which leaves it bound to the protection measures applied to this information. Vulnerable STP allows a malicious user to control and define the path the mechanism would take; this allow the attacker to hide nodes from the mechanism, or even cause a denial of service attack by adding network loops to the topology. Notice that this is not a direct attack on the mechanism itself but on the feedback the scanning mechanism uses to function. The same apply to the CAM table, what if a malicious user alters it to reflect no switches available; this would lead the mechanism to find no further way to propagate and stop. More threats exist in the backbone as well. The scanning mechanism depends on OSPF protocol to pass to other LANs, what if an attacker was able to forge the LSD to contain non-existent IP addresses, which will stop the vulnerability discovery process from reaching further LANs. As a matter of fact, the compromise of the LSD would allow the attacker to define routers IP addresses that is not necessary part of of the predefined scope of the mechanism. Likewise with the ARP cash stored at routers, a poisoned ARP cache can give the attacker the ability to direct the worm to self-replicate to whatever IP is given as long the target is vulnerable. This would take the scanning mechanism out of its predefined scope and would add a malicious intention to its operational procedure. Detecting new switches and nodes is also threatened by malicious activity. One scenario would be adding a rogue switch where the scanning mechanism would not be able to deal with. When the mechanism probes the rogue switch for CAM or STP information, the switch would provide misleading response, such as fake designate Bridge, allowing the switch to remain with its directly connected malicious nodes away from the mechanism coverage. Our mechanism assumes the topology information to be authentic and hasn't been altered. However, if that is not the case, it becomes vulnerable to the same vulnerabilities the network topology has. This is expected as the mechanism in its topology dependent nature becomes part of the network, like the protocols operating within the network, not like an independent external security system. The simulation results cover different aspects of performance. Node operation and link failures of $p = 0.05$ and $p = 0.01$, respectively, have recorded different results. For example in a network of size 2500 nodes, link failures were 27 and node operation failures were 131. Further, 85 link failures were recorded at a network of size 8000 nodes and 420 node operation failures at the same network. Fig. 3 shows more results. The time it took the agent to cover a network of 2500 elements was 3.25 simulation seconds and 4.0s to cover a 5000 network topology. Of course the time is affected by link and node operation failures, the improved mechanism will wait 0.5s before retrying after a link or node operation failure. Note that the time reflects the coverage of the whole network including newly added network devices, that join the network at simulation second 2.5. Fig. 4 shows more results. For the topology change detection, which records the time it takes the propagation mechanism to detect all newly added network devices, it shows close results. That's because each topology detects almost the same number of network devices. In a network of size 3000 it required 0.47s to detect 30 hosts and a switch (with different hosts connected to it) and 0.48s of

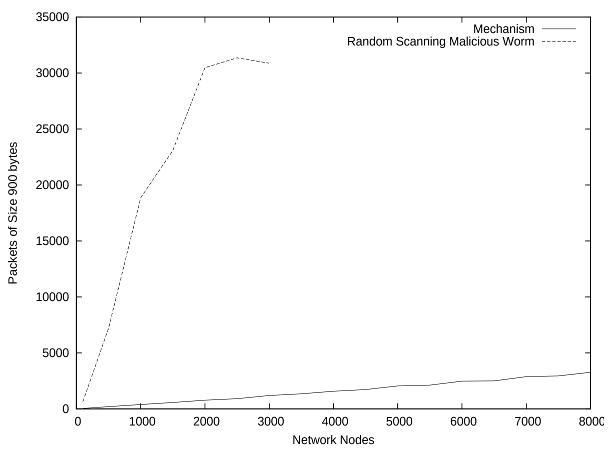

Fig. 6. Number of packets generated by randomly scanning worm in comparison to our scanning mechanism

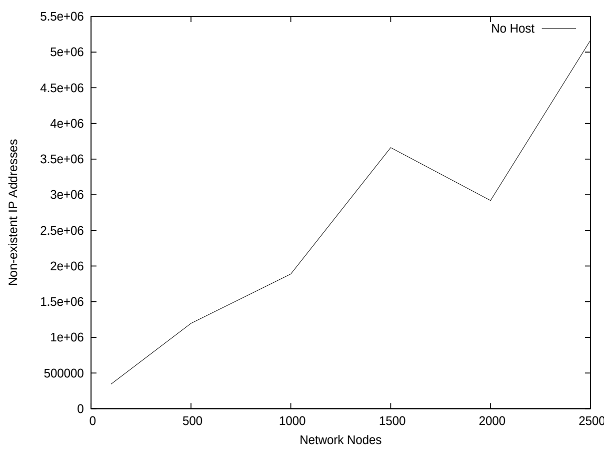

Fig. 7. The random scanning worm number of scan attempts of non-existent IP addresses

a network of size 8000. More results can be seen at Fig. 5. In comparison with a randomly scanning worm, our mechanism has generated 570 packets in a network of size 1500, while the random scanning worm generated 23074 packets. Moreover, the random worm generated 31358 packets to cover a network of size 2500 while our scanning algorithm required 912 packets only. Further results are shown in Fig. 6. Also, the random scanning worm has generated too many scans to non-existent IP addresses. Although the worm operates within the LAN, non-existent IP scans will cause the host to emit an ARP probe for each new IP address scanning attempt. For a network topology of size 1500 it resulted in 3660723 non-existent IP scan attempts and 5165367 for a network of size 2500. Results are shown in Fig. 7. Our mechanism, however, avoid scanning non-existent IP addresses since it detect targets based on CAM information.

XIII. Conclusion

In this paper we have highlighted the need for effective mechanisms to detect network nodes vulnerabilities. We have previously proposed an agent-based mechanism to help prevent malicious attacks, and in this paper we further enhance its performance. Improvements include: edge node failure recovery to ensure sound and efficient coverage of all vulnerable edge nodes and an algorithm to enable the mechanism to traverse the network backbone using OSPF protocol and ARP stored in routers – which provides independently driven self-propagation to all network LANs without human intervention as compared to the previous design. We also proposed an algorithm to enable the mechanism to detect newly added network devices as soon as possible to eliminate vulnerability exposure. Since the mechanism is topology dependent, different risks and threats to topology information have been highlighted, especially protocols utilized in the discovery process, such as: STP, OSPF, ARP, SNMP, and CAM. We have validated our mechanism through simulations; the results shown here assumed relatively volatile link failures of probability $p = 0.01$ and node operation failure of $p = 0.05$. The results also showed the time it took the mechanism to cover the network, in addition to the time required to detect newly vulnerable nodes that have just joined the corporate network. Simulation results of a randomly scanning worm that to some extent mimic the propagation behavior of the Slammer worm has also been gathered to be compared with the mechanism in terms of bandwidth utilization. Ongoing and future work is focused on increasing both the robustness and performance of the algorithms in defending the propagation mechanism against malicious adversaries.

Acknowledgments: This research has been funded by Prince Muqrin Chair of IT Security (PMC) at King Saud University.

References

[1] "Annual Report Pandalabs 2009," Panda Security, Tech. Rep., 2010.

[2] P. Porras, H. Saidi, and V. Yegneswaran, "An Analysis of Conficker's Logic and Rendezvous Points," SRI International, Tech. Rep., 2009.

[3] ——, "Conficker C Analysis," SRI International, Tech. Rep., 2009.

[4] D. Aitel, "Nematodes beneficial worms." 2006, [Online; accessed 13-April-2010]. [Online]. Available: http://www.immunityinc.com/downloads/nematodes.pdf

[5] J. Aspnes, N. Rustagi, and J. Saia, "Worm Versus Alert: Who Wins in a Battle for Control of a Large-Scale Network?" in *Proceedings of the 11th International Conference on Principles of Distributed Systems (OPODIS 2007)*. Springer-Verlag.

[6] Z. Al-Salloum and S. Wolthusen, "Link-Layer-Based Self-Replicating Vulnerability Discovery Agent," in *Proceedings of the Fifteenth IEEE Symposium on Computer and Communications*. IEEE Press, 2010.

[7] E. H. Spafford, "The Internet Worm Program: An Analysis," Department of Computer Sciences, Purdue University, West Lafayette, IN, USA, Tech. Rep. CSD-TR-823, Dec. 1988.

[8] S. Staniford, V. Paxson, and N. Weaver, "How to own the internet in your spare time," in *Proceedings of the 11th USENIX Security Symposium*. Berkeley, CA, USA: USENIX Association, 2002, pp. 149–167.

[9] X. Fan and Y. Xiang, "Accelerating the Propagation of Active Worms by Employing Multiple Target Discovery Techniques," in *Proceedings of the IFIP International Conference on Network and Parallel Computing (NPC 2008)*. Springer-Verlag.

[10] J. Nazario, *Defense and Detection Strategies against Internet Worms*. Artech House Publishers, October 2003.

[11] C. C. Zou, D. Towsley, W. Gong, and S. Cai, "Routing worm: A fast, selective attack worm based on ip address information," in *PADS '05: Proceedings of the 19th Workshop on Principles of Advanced and Distributed Simulation*. Washington, DC, USA: IEEE Computer Society, 2005, pp. 199–206.

[12] C. C. Zou, D. Towsley, and W. Gong, "On the performance of internet worm scanning strategies," *Elsevier Journal of Performance Evaluation*, vol. 63, pp. 700–723, 2003.

[13] D. Moore, C. Shannon, and J. Brown, "Code-red: a case study on the spread and victims of an internet worm," in *Proceedings of the ACM SIGCOMM / USENIX Internet Measurement Workshop (IMW 2002)*, Marseille, France, 2002, pp. 273–284.

[14] M. Vojnović, V. Gupta, T. Karagiannis, and C. Gkantsidis, "Sampling Strategies for Epidemic-Style Information Dissemination," in *Proceedings of the 27th IEEE Conference on Computer Communication (INFOCOM 2008)*. Phoenix, AZ, USA: IEEE Press, Apr. 2008, pp. 1678–1686.

[15] T. Simonite, "Friendly 'worms' could spread software fixes," *NewScientist*, 2008, [Online; accessed 13-April-2010]. [Online]. Available: http://www.newscientist.com/article/dn13318

[16] M. Kern, "Re: Codegreen beta release (idq-patcher/anticodered/etc.)," 2001, [Online; accessed 13-April-2010]. [Online]. Available: http://seclists.org/vuln-dev/2001/Sep/0001.html

[17] F. Leder and T. Werner, "Know Your Enemy: Containing Conficker," The Honeynet Project, University of Bonn, Germany, Tech. Rep., Mar. 2009.

[18] Z. Al-Salloum and S. Wolthusen, "Semi-Autonomous Link Layer Vulnerability Discovery and Mitigation Dissemination," in *Proceedings of the 5th International Conference on IT Incident Management & IT Forensics (IMF 2009)*. IEEE Press, 2009.

[19] J. T. Moy, *OSPF: Anatomy of an Internet Routing Protocol*. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 1998.

[20] Y. Breitbart, M. Garofalakis, B. Jai, C. Martin, R. Rastogi, and A. Silberschatz, "Topology discovery in heterogeneous ip networks: the netinventory system," *IEEE/ACM Trans. Netw.*, vol. 12, no. 3, pp. 401–414, 2004.

[21] R. Seifert and J. Edwards, *The All-New Switch Book: The Complete Guide to LAN Switching Technology*. Wiley Publishing, 2008.

[22] J. Nazario, J. Anderson, R. Walsh, and C. Connelly, "The future of internet worms," 2001.

[23] D. Moore, V. Paxson, S. Savage, C. Shannon, S. Staniford, and N. Weaver, "Inside the slammer worm," *IEEE Security and Privacy*, vol. 1, no. 4, pp. 33–39, July 2003.

[24] E. Seagren, *Secure Your Network for Free*. Syngress Publishing, 2007.

[25] Z. Al-Salloum and S. Wolthusen, "Security and Performance Aspects of an Agent-Based Link-Layer Vulnerability Discovery Mechanism," in *Proceedings of the Fourth International ARES Workshop on Secure Software Engineering (SecSE 2010)*. IEEE Press, 2010.

[26] T. Issariyakul and E. Hossain, *Introduction to Network Simulator NS2*. Heidelberg, Germany: Springer-Verlag, 2009.

[27] E. Skoudis and L. Zeltser, *Malware: Fighting Malicious Code*. Prentice Hall PTR, November 2003.