

Defending against Hitlist Worms using Network Address Space Randomization

K. G. Anagnostakis[†], S. Antonatos^{*}, P. Akritidis^{*}, E. P. Markatos^{*}

[†]Distributed Systems Laboratory
CIS Department, Univ. of Pennsylvania
200 S. 33rd Street, Philadelphia, PA 19104, USA
anagnost@dsl.cis.upenn.edu

^{*} Institute of Computer Science
Foundation for Research & Technology – Hellas
P.O.Box 1385 Heraklio, GR-711-10 GREECE
{antonat,akritid,markatos}@ics.forth.gr

Abstract

Worms, self-replicating programs which infect vulnerable hosts on the Internet, are a major security threat. Sophisticated worms that use precomputed hitlists of vulnerable targets are especially hard to contain, as they spread at rates where existing defenses may not be able to act in a timely fashion and have characteristics that make them more difficult to detect. This paper proposes a new approach, called Network Address Space Randomization (NASR), whose goal is to harden networks specifically against hitlist worms. The idea behind NASR is that hitlist information could become stale if nodes are forced to change their IP addresses frequently enough. NASR slows down hitlist worms, and could even force them to exhibit scan-like features that makes them easier to contain at the perimeter. We explore the design space for NASR and present a prototype implementation as well as preliminary experiments analyzing the effectiveness of the approach.

1 Introduction

Worms and viruses are widely regarded to be one of the major security threats facing the Internet today. Incidents such as Code Red[1, 15] and Slammer[3] have clearly demonstrated that worms can infect tens of thousands of computers in less than half an hour, a timescale where human intervention is unlikely to be feasible. More recent research studies have estimated that worms can infect as many as a million hosts in less than two seconds [20, 21, 22]. Unlike most of the currently known worms that find their victims by targeting random IP addresses in search for vulnerable hosts, these extremely fast worms rely on *hitlists*, pre-computed lists of vulnerable targets, in order to spread

efficiently.

The threat of worms and the speed at which they can spread have motivated research in automated worm detection and defense mechanisms. For instance, several recent studies have focused on detecting scanning worms. Techniques such as [25, 12, 24, 17, 19, 23] detect scanning activity and either block or throttle further connection attempts. These techniques are unlikely to be effective against hitlist worms, as they do not exhibit the failed-connection feature that scan detection is looking for. To improve the effectiveness of worm detection, several distributed, cooperative defense and early-warning systems have been proposed, with the goal of aggregating scanning or other indications of worm activity from different sites [26, 16, 8, 27]. Distributed detection is usually slower, as it requires data collection and correlation among different sites, and is unlikely to be able to detect an attack at the estimated timescales of hitlist worms.

This paper considers the question of whether it is possible to defend against hitlist worms. We first examine strategies for building hitlists and how effective these strategies can be. We observe that hitlists tend to *decay* naturally for various reasons, as hosts get disconnected or change addresses, and applications are started and shut down. A rapidly decaying hitlist is likely to decrease the spread rate of a worm. It may also increase the number of unsuccessful connections it initiates, and may thus increase exposure of the worm to scan-detection methods.

Starting with this observation about hitlist decay, it is natural to ask if it is possible to *intentionally* induce hitlist decay, and we examine the possibility of achieving this through *network address space randomization* (NASR). This technique is inspired by instruction address randomization that has been proposed to protect against code injection attacks at the compiler level[13]. It is also similar in principle to the “IP hopping” mech-

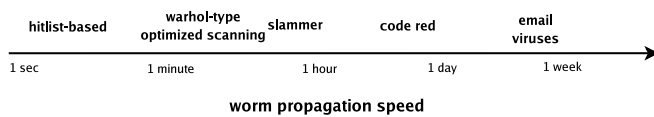


Figure 1. Propagation speed and severity of different types of attacks

anism presented in [9], whose goal is to confuse targeted attacks. In this paper, we apply the same basic idea to the specific context of defending against hitlist worms. In its simplest form, network address space randomization can be provided by adapting dynamic IP address allocation services such as DHCP¹ to *force* more frequent address changes. This simple approach may be able to protect enabled networks against hitlist worms, and, if deployed at a large enough scale, may be able to significantly hamper their spread.

We must emphasize that, like most (if not all) other worm containment proposals, network address space randomization is only a partial solution to the worm containment problem. The basic advantage of our approach is the ability to slow down, rather than completely squash hitlist-based worm epidemics. Slowing down the fastest known method propagation is important as it may allow more time for other, possibly cooperative defenses to kick in. Furthermore, we must note that our analysis does not invalidate the worst-case estimates provided in previous work, nor is our goal to play down the threat posed by such worms. Rather, we show that network address space randomization can provide a useful improvement at reasonable cost, for certain types of attacks, and is thus worth considering as part of a broader worm defense portfolio.

The rest of this paper is organized as follows. In Section 2 we provide some background on worms, hitlists, and relevant detection mechanisms. In Section 3 we explore in more detail the idea of network address space randomization, and outline a randomized DHCP server implementation. In Section 4 we analyze various hitlist generation strategies, and present measurements exploring the properties of a small subset of the IP address space. In Section 4 we present a simulation study analyzing the effectiveness of network address space randomization in terms of how much it would slow down a hitlist worm and how it would expose such a worm to scan detection. We summarize our results and conclude in Section 5.

¹Another known address allocation service is `bootp`, but it allocates addresses semi-permanently, without any mechanism for renewing the allocation, and is thus not usable for our purposes.

2 Background

For the purpose of placing our work in context, we first give a brief overview of what is known about worms, with some emphasis on hitlist worms, and present a summary of proposals for defending against worms and how they relate to hitlist worms which are the focus of this paper.

2.1 Worms

Computer viruses have been studied extensively over the last couple of decades. Cohen was the first to define and describe computer viruses in their present form. In [11], he gave a theoretical basis for the spread of computer viruses. The strong analogy between biological and computer viruses led Kephart *et al.* [14] to investigate the propagation of computer viruses based on epidemiological models. They extend the standard epidemiological model by placing it on a directed graph, and use a combination of analysis and simulation to study its behavior. They conclude that if the rate at which defense mechanisms detect and remove viruses is sufficiently high relative to the rate at which viruses spread, it is possible to prevent widespread virus propagation.

The Code Red worm [1] was analyzed extensively in [28]. The authors conclude that even though epidemic models can be used to study the behavior of Internet worms, they are not accurate enough because they cannot capture some specific properties of the environment these operate in: the effect of human countermeasures against worm spreading (*i.e.*, patching, filtering, disconnecting, *etc.*), and the slowing down of the worm infection rate due to the worm’s impact on Internet traffic and infrastructure. They derive a new general Internet worm model called *two-factor worm* model, which they then validate in simulations that match the observed Code Red data available to them. Their analysis seems also to be independently supported by the data on Code Red propagation in [15].

A similar analysis on the SQL “Slammer” (or Sapphire) worm [2] can be found in [3]. Sapphire, the fastest worm to day, was able to infect more than 70,000 victim computers in less than 15 minutes.

The Blaster/Welchia epidemic is an interesting example of a “vigilante” worm (Welchia) causing more trouble than the original outbreak (Blaster). A “vigilante” worm attempts to clean-up another worm by using the same vulnerability. However, the very notion of “vigilante” worms is rendered useless if worms immediately disable the vulnerability after compromis-

ing a machine.

The Witty worm [18] is of interest for several reasons. First, it was the first widely propagated Internet worm to carry a destructive payload. Second, Witty was started in an organized manner with an order of magnitude more ground-zero hosts than any previous worm and also began to spread as early as only one day after the vulnerability was publicized, which is an indication that the worm authors had already prepared all the worm infrastructure, including the ground-zero hosts and the replication mechanisms, and were only waiting for an exploit to become available in order to launch the worm. Finally, Witty spread through a population almost an order of magnitude smaller than that of previous worms, showing that a hitlist is not required even for targeting small populations.

All these worms use (random) scanning to determine their victims, by using a random number generator to select addresses from the entire IP address space. Although some worms chose their next target uniformly among all the available IP addresses, other worms seemed to prefer local addresses over distant ones, so as to spread the worm to as many local computers as possible. Once inside an organization, these worms make sure that they will infect several of the organization's computers before trying to infect any outside hosts.

2.2 Hitlists

Instead of attempting to infect random targets, a worm could first determine a large vulnerable population before it starts spreading. The worm creator can assemble a list of potentially vulnerable machines prior to releasing the worm, for example, through a slow port scan. The list of known vulnerable hosts is called a hitlist. Using hitlists, worms do not need to waste time scanning for potential targets during the time of the attack, and will not generate as many unsuccessful connections as when scanning randomly. This allows them to spread much faster, and it also makes them less visible to scan-based worm detection tools. A hitlist can be either a collection of IP addresses, a set of DNS names or a set of Distributed Hash Table identities (for infecting DHT systems irrelevantly of the network infrastructure).

Hitlist worms have not been observed in the wild, perhaps because the co-evolution of worms and defenses has not reached that stage yet: they are not currently *necessary* for a successful worm epidemic, since neither scan-blocking nor distributed detection systems are widely deployed yet. However, hitlists

are certainly feasible today and worm creators are very likely to use them in the future.

Hitlist worms have attracted some attention lately, as they are easy to model off-line [21, 20]. In this context, several hitlist construction methods have been outlined: random scanning, DNS searches, web crawling, public surveys and indexes, as well as monitoring of control messages in peer-to-peer networks.

There are many ways for building hitlists. Random scanning can be used to compile a list of IP addresses that respond to active probing. Since the addresses will not be (ab)used immediately, the worm author can use so-called stealth, low rate, scanning techniques to make the scan pass unnoticed. On the other hand, if the duration of the low-rate scanning phase is very long, some IP addresses will eventually expire.

Hitlists of Web servers can be assembled by sending queries to search engines and by harvesting Web server names off the replies. Similar single-word queries can also be sent to DNS servers in order to validate web server names and find their IP addresses. Interestingly enough, these types of scans can be used to easily create large lists of web servers, and are very likely to go unnoticed.

However, any form of active scanning, probing, or searching, has the potential risk of being detected. This gives special appeal to passive techniques, such as those based on peer-to-peer networks. Such networks typically advertise many of their nodes and this information can be collected by just observing the traffic that is routed through a peer. The creation of the hitlist does not require any active operation from the peer-to-peer node and therefore cannot raise suspicion easily.

2.3 Worm defenses

We discuss some recent proposals for defending against worms and whether they could be effective against hitlist worms.

Approaches such as the one by Wu *et al.* [25] attempt to detect worms by monitoring unsolicited probes to unassigned IP addresses (“dark space”) or inactive ports. Worms can be detected by observing statistical properties of scan traffic, such as the number of source/destination addresses and the volume of the captured traffic. By measuring the increase on the number of source addresses seen in a unit of time, it is possible to infer the existence of a new worm when as little as 4% of the vulnerable machines have been infected.

An approach for isolating infected nodes inside an

enterprise network is discussed in [19, 12]. The authors show that as little as 4 probes may be sufficient in detecting a new port-scanning worm. Weaver *et al.* [23] describe a practical approximation algorithm for quickly detecting scanning activity that can be efficiently implemented in hardware. Schechter *et al.* [17] use a combination of reverse sequential hypothesis testing and credit-based connection throttling to quickly detect and quarantine local infected hosts. These systems are effective only against scanning worms (not topological, or “hit-list” worms), and rely on the assumption that most scans will result in non-connections.

Several cooperative, distributed defense systems have been proposed. DOMINO is an overlay system for cooperative intrusion detection [26]. The system is organized in two layers, with a small core of trusted nodes and a larger collection of nodes connected to the core. The experimental analysis demonstrates that a coordinated approach has the potential of providing early warning for large-scale attacks while reducing potential false alarms. Zou *et al.* [27] describes an architecture and models for an early warning system, where the participating nodes/routers propagate alarm reports towards a centralized site for analysis. The question of how to respond to alerts is not addressed, and, similar to DOMINO, the use of a centralized collection and analysis facility is weak against worms attacking the early warning infrastructure. Fully distributed defense mechanisms, such as [16, 8] may be more robust against infrastructure attacks, yet all distributed defense mechanisms that we are aware of are likely to be too slow for the estimated timescales of hitlist worms.

3 Network Address Space Randomization

The goal of network address space randomization (NASR) is to force hosts to change their IP addresses frequently enough so that the information gathered in hitlists is rendered stale by the time the hitlist-based worm is unleashed.

To illustrate the basic idea more formally, consider an abstract system model, with an address space $R = \{1, 2, \dots, n\}$, a set of hosts $H = \{h_1, \dots, h_m\}$ where $m < n$, and a function A that maps all hosts h_k to addresses $A(h_k) = r \in R$. Assume that at time t_a , the attacker can (instantly) generate a hitlist $X \subset R$ containing the addresses of hosts that are live and vulnerable at that time. If the attack is started at time t_x and all hosts in X are still live and vulnerable and have the same address as at time t_a then the worm can very

quickly infect $|X|$ hosts.

In a system implementing NASR, consider that at time t_b where $t_a < t_b < t_x$, all hosts are assigned a new address from R . Thus, at the time of the attack t_x the probability that a hitlist entry x_k still corresponds to a live host is $p = m/n$ and thus the attacker will be able to infect $(m/n)|X|$ hosts. Besides reducing the number of successfully infected nodes in the hitlist, the attack will also result in a fraction $1 - m/n$ of all attempts failing (which may be detectable using existing techniques). In this simple model, the density m/n of the address space seems to be a crucial factor in determining the effectiveness of NASR. So far we have assumed a homogeneous set of nodes, all with the same vulnerability and probability of getting infected. If only a subset of the host population is vulnerable to a certain type of attack, then the effectiveness of NASR in reducing the fraction of infected hitlist nodes and the number of failed attempts is proportionally higher.

3.1 Practical considerations

The model we presented illustrates the basic intuition of how NASR can affect a hitlist worm. Mapping the idea to the reality of existing networks requires us to look into several practical issues.

First, random assignment of an address from a global IP address space pool is not practical for several reasons: (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, such as those based on access lists, and (iii) it requires global coordination for being implemented and is thus less practical. The difficulty of implementing NASR decreases as we restrict its scope to more local regions. Each AS could implement AS- or prefix-level NASR, but this would still create administrative difficulties with interior routing and access lists. It seems that a reasonable strategy would be to provide NASR at the subnet-level, although this does not completely remove the problems outlined above. For instance, access lists would need to be reconfigured to operate on DNS names and DNS would need to be dynamically updated when hosts change addresses.

Second, some nodes cannot change addresses and those that can may not be able to do so as frequently as we would want. The reason for this is that addresses have first-class transport- and application-level semantics. For instance, DNS server addresses are usually hardcoded in system configurations. Even for

DHCP-configured hosts, changing the address of a DNS server would require synchronizing the lease durations so that the DNS server can change its address at exactly the same time when *all* hosts refresh their DHCP leases. While technically feasible, this seems too complex to implement and such complexity should rather be avoided. Similar constraints hold for routers.

Generally, all active TCP connections on a host that changes its address would be killed, unless connection migration techniques such as [24, 10] are used. Such techniques are not widely deployed yet and it is unrealistic to expect that they will be deployed soon enough to be usable for the purposes of NASR. Fortunately, many applications are designed to deal with occasional connectivity loss by automatically reconnecting and recovering from failure. For such applications, we can assume that infrequent address changes can be tolerated. Examples of these applications are many P2P clients, like Kazaa and DirectConnect, Windows/SAMBA sharing (when names are used), messengers, chat clients, etc. However, tolerance does not always come for free: frequent address changes may result in churn in DHT-based applications, and would generally have the side-effect of increasing stale state in other distributed applications, including P2P indexing and Gnutella-like host caches. Finally, some applications are even less tolerant to failures. For instance, NFS clients often hang when the server is lost, and do not transparently re-resolve the NFS server address from DNS before reconnecting.

There exist ways to make systems more robust to address changes. Rocks [10] is one solution providing reliable sockets for protecting applications sensitive to IP address changes. However, it must be present at both ends of the connection, so it is not practical for connections with external third parties. In a LAN environment, a similar solution using a “reverse NAT” box may be applicable in some cases, with the client host being oblivious to address changes, and the NAT middlebox making sure that address changes do not affect applications. However, this too seems to require an infrastructure overhaul that we would prefer to avoid.

All these practical constraints suggest that NASR should be implemented very carefully. A plausible scenario would involve NSR at the subnet level, and particularly for client hosts in DHCP-managed address pools. How such concessions affect NASR, as well as the rate at which address changes should be made for NASR to be effective will be explored in more detail in Sections 4 and 5.

3.2 Implementation

A basic form of address space randomization can be implemented using minor modifications to a DHCP server. The server must be configured to expire DHCP leases at intervals suitable for effective randomization. The server would normally allow hosts to renew the lease if hosts request that before the lease expires. To force addresses changes even if hosts request to renew the lease before it expires, a new option, *iprand-interval*, is needed. This option specifies how frequently an IP address change must be enforced. The only essential modification necessary is to have the DHCP server consider *iprand-interval* to decide between renewing the lease and replacing it with a new one, on a different address.

Several improvements are possible, to make randomization more effective and reduce the risk of adverse effects on applications. We consider two features for our implementation. First, we avoid assigning an address to a host that has significant overlap in services (and potential vulnerabilities) with hosts that have used the address before. Second, we can try to avoid forcing address changes for hosts that have active connections or a given service profile.

We have implemented an advanced randomization-enabled DHCP server based on the standard open-source DHCP implementation. Our extension provides *activity monitoring* and *service fingerprinting*.

Activity monitoring keeps track of open connections and tries to avoid forcing an address change on a host whose services could be disrupted. In our prototype, we only consider long-lived TCP connections (that could be, for example, FTP downloads). More complicated policies can be implemented, but are outside the scope of our proof-of-concept implementation.

Service fingerprinting examines traffic on the network and attempts to identify what services are running on each host. For instance, a TCP connection to port 80 suggests that the host is running a Web server. During address allocation, the list of running services is considered so as not to perform randomization between hosts running the same services. Port number and other activity may allow guessing the operating system of a host. For instance, port 445 is an indication that a host might be a Windows platform. Randomization between hosts with different operating systems, e.g., between a Windows and a Linux platform seems to be a good strategy. Although our implementation of these features is rudimentary, there are many documented techniques for fingerprinting, some of which

are available as part of open-source tools[7, 6, 5, 4].

4 Measurements

To explore the design space of network address space randomization we first need to consider some basic hitlist characteristics, such as the speed at which a hitlist can be constructed, the rate at which addresses already change (without any form of randomization), and how address space is allocated and utilized. We perform measurements on the Internet to obtain a clearer picture of these characteristics.

4.1 Random scanning

We determine the effectiveness of random scanning for building hitlists. We first generate a list of all /16 prefixes that have a valid entry with the `whois` service, in order to increase scan success rates and avoid reserved address space. We then probe random targets within those prefixes using ICMP ECHO messages. Using this approach, we generated a hitlist of 20,000 addresses. Given this hitlist, we probe each target in the hitlist once every hour for a period of two weeks. Every probe consists of four ICMP ECHO messages spaced out over the one-hour run in order to reduce the probability of accidentally declaring an entry stale because of short-term congestion or connectivity problems. Note that these measurements do not give us exactly the probability of the worm successfully infecting the target host, but only a rough estimate. Although we were tempted to perform more insightful reconnaissance probes on the nodes in the hitlist, this would result in a much higher cost in terms of traffic and a high risk of causing (false) alarms at the target networks. The techniques would most likely involve full port scans, application-level fingerprinting and more frequent probes to perform `ipid`-based detection of host changes.

The results of the ICMP ECHO experiment are shown in Figure 2. We observe that the hitlist decays rapidly during the first day, and continues to decay, albeit very slowly, over the rest of the two-week run. The number of reachable nodes tends to vary during the time of day, apparently peaking on business hours in the US with minor peaks that may coincide with working hours elsewhere in the world. Overall, the decay of the hitlist slows down over time, reaching an almost stable level of 75% of hitlist nodes reachable.

4.2 Passive P2P snooping

In the Gnutella P2P network, node addresses are carried in QueryHit and Pong messages. A Gnutella client can harvest thousands of addresses without performing any atypical operations. In our experiments, a 24-hour period sufficed for gathering 200K unique IP addresses, as shown in Figure 5. Intensive searches and using other, more popular, P2P networks will probably result in higher yield.

Most P2P nodes are short-lived. Addresses harvested through P2P networks become unavailable very quickly. Figure 3 shows the decay of the hitlist as a function of elapsed time. Note that in this experiment we only check whether the nodes respond to ICMP ECHO probes, not whether the Gnutella client is still up and running. Thus, it is possible that the IP address is not used by the same host recorded in the hitlist. This may or may not be important for the attacker, depending on how much the attack depends on software versions and whether version information has been used in constructing the hitlist.

4.3 Search-engine harvesting

Querying a popular search engine for *the* or similar keywords returns hundreds of millions of results. Retrieving a thousand results gave 612 unique alive hosts and 30 dead hosts. Most search engines restrict the number of results that can be retrieved, but the attacker can use multiple keywords, either randomly generated or taken from a dictionary.

The hosts that immediately appear as dead are a result of the frequency of the indexing by the search engine. It plays a role in the speed of harvesting the addresses and must be considered for the decay if the addresses are not checked.

Figure 4 shows the decay of the hitlist created using the search engine results. We observe that, compared to the other address sources, the search engine results are very stable. This was expected, since web servers have to be online and use stable addresses. It does not mean, however, that addresses retrieved through search engines are better suited for attackers. Depending on the vulnerability at hand, unprotected, client PCs, such as those returned by crawling peer-to-peer networks may be preferred.

A natural question to ask is whether such suspicious behavior can be detected easily by the search engines. This seems unlikely: an attacker motivated enough to unleash a worm is likely to also be capable of setting up a botnet (a set of machines under the control of the

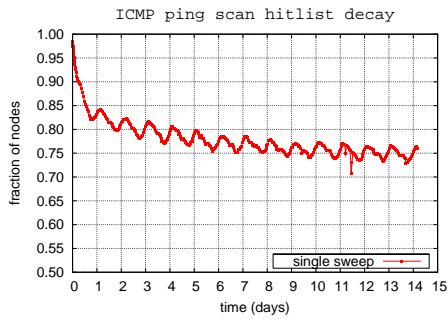


Figure 2. Decay of addresses harvested using random scanning

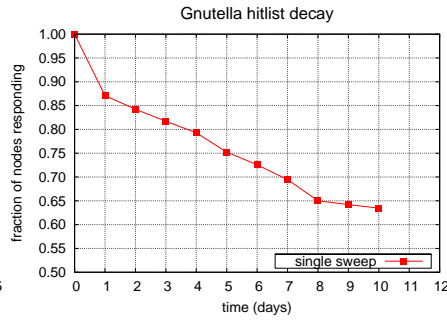


Figure 3. Decay of addresses harvested by monitoring peer-to-peer traffic routed through a node.

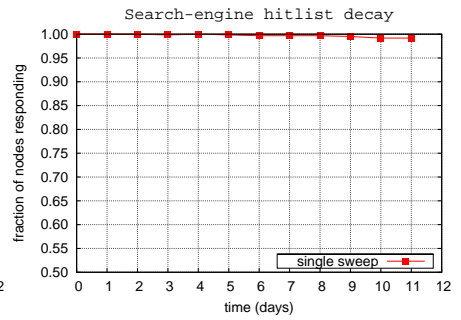


Figure 4. Decay of addresses harvested by querying a popular web search engine.

attacker) and issuing multiple queries from different locations in a way that can make detection hard while maximizing the aggregate rate of retrieving hosts addresses.

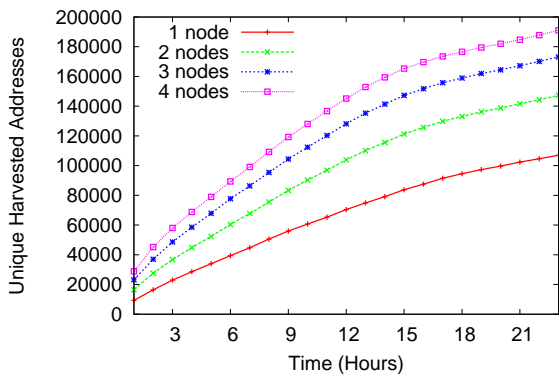


Figure 5. Number of distinct addresses harvested by monitoring Gnutella traffic as a function of time and number of monitoring nodes.

4.4 Subnet address space utilization

The feasibility and effectiveness of network address space randomization depend on how many unused addresses there are in NASR-enabled subnets. Performing randomization on a subnet with many unused addresses will result in the worm failing to connect to a hitlist target with higher probability than on a highly utilized subnet. Such failures could expose the worm as they could be picked up by scan-detection mechanisms. If the subnet is highly utilized, and if we

assume a homogeneous network with identical hosts (e.g., running the same services) then the worm is more likely to succeed in infecting a host, even if the original host recorded in the hitlist has actually changed its address. Finally, in the extreme (and most likely rare) case of a subnet that is always fully utilized, then there will never be a free slot to change a node's address.

We attempt to get an estimate of typical subnet utilization levels. Because of the high scanning activity, we cannot perform this experiment on a global scale without tripping a large number of IDS alerts. We therefore opted for scanning five /16 prefixes that belong to FORTH, the University of Crete, and a large ISP, after first explaining the nature of the experiment and obtaining permission by the administrators of the networks. We performed hourly scans on all prefixes using ICMP ECHO messages over a period of one month. For simplicity, we assume that all prefixes are subnetted in /24's.

A summary of the results is shown in Figure 7. We see that many subnets were completely dark with no hosts at all (not even a router). Nearly 30% of the subnets in two ISP prefixes were totally empty, while for the FORTH and UoC the percentage reaches 70%. This means that swapping subnets would likely be an effective NASR policy, but unfortunately it is not practical, as discussed previously. We also see that 95% of these subnets have less than 50% utilization and the number of maximum alive hosts observed does not exceed one hundred. If subnet utilization at the global level are similar to what we see in our limited experiment, NASR at the level of /24 subnets is likely to be effective, as there is sufficient room to move hosts around, reducing the effectiveness of the worm and causing it to make failed connections.

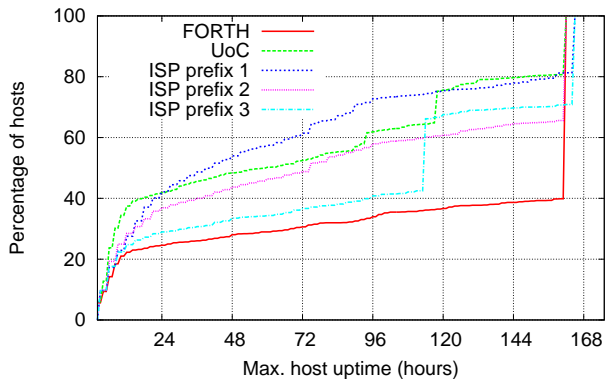


Figure 6. Host uptime distribution

4.5 Host uptime measurements

Aiming at better understanding the behavior of the hosts, we measured the maximum uptime for the hosts that were at least one time alive. These measurements will tell us whether hosts disconnect frequently enough so that their address can be changed between disconnects, and not while they are actively using the network.

The cumulative distribution function is shown in Figure 6. The liveness of the hosts was monitored for a full week by sending ping messages every hour. Almost 60% of the hosts inside FORTH were always up, which seems reasonable as it is an environment consisting mostly of workstations. In more dynamic environments, like the ISP and University of Crete networks only 20-30% of the hosts were consistently alive, while nearly 40% of the hosts were alive for maximum 10 hours. Although such dynamic environments perform some form of natural randomization on their address space, mostly due to DHCP, most of the DHCP servers are configured to maintain leases for machines connecting to the network. The usual scenario is that a DHCP server is giving the same IP to a specific host (by caching its Ethernet address). Typically, a lease expires in 15 days period, so hosts that do not refresh the lease before it expires (e.g., because they are not connected) would obtain a new address. Although we do not have measurements on how often this happens, it seems likely that this minor, slow form of randomization is unlikely to be effective by itself.

5 Simulation study

It is infeasible to run experiments on the scale of the global Internet. To evaluate the effectiveness of our

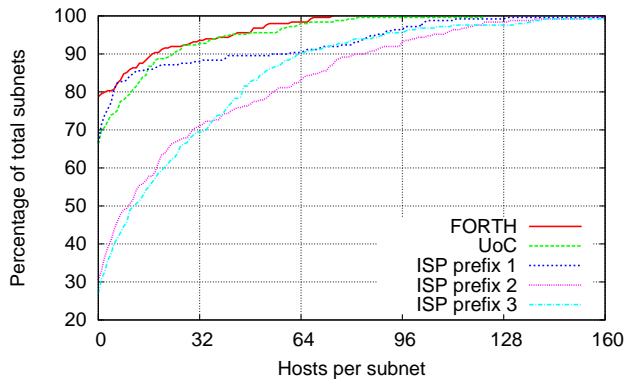


Figure 7. Subnet address space utilization

design, we simulated a small-scale (compared to the Internet) network of 1,000,000 hosts, each of which could be a potential target of worms.

Because of the variety of operating systems used and services provided, we assume that a fraction of hosts v is vulnerable to the worm. For simplicity, we ignore the details of the network topology, including the effect of end-to-end delays and traffic generated by the worm outbreak. We simply consider a flat topology of routers, each serving a subnet of end-hosts.

A fraction of addresses is allocated in each subnet, which affects the probability of successful scan attempts within the subnet. This probability is an important parameter in the case where a host in the hitlist has changed its address, because it determines if *another* live host would be available at the same address. A separate parameter is used for random scanning, reflecting the fraction of the overall address space that is completely unused.

The hitlist is generated at configurable rates, and we assume that the worm starts spreading immediately after finishing with generating the hitlist. Because the early hitlist entries are more likely to have become stale between their discovery and the start of the attack, the worm starts attacking the freshest addresses in the hitlist first. For simplicity, we ignore the details of how the hitlist is distributed and encoded in the payload of the worm: we assume that every worm instance can obtain the next available entry at zero cost. After finishing with the hitlist, we assume that the worm will continue trying to infect hosts using random scanning.

5.1 Impact of NASR

In the first experiment, we simulate work outbreaks with different parameters, and measure the worm

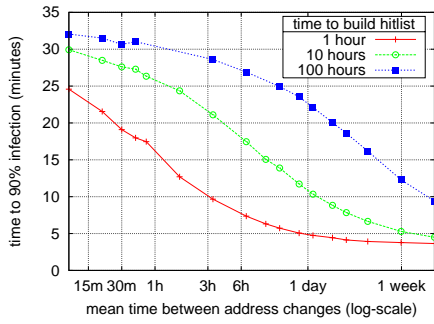


Figure 8. Worm spread time (time to 90% infection) vs. time between host address changes for different hitlist generation rates

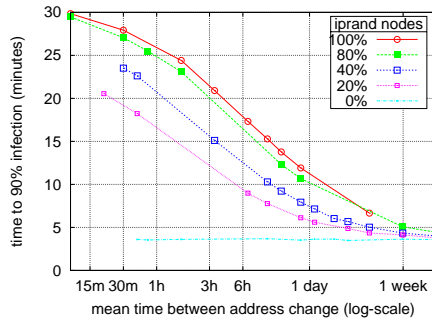


Figure 9. Effect of network address space randomization on worm spread time when partially deployed

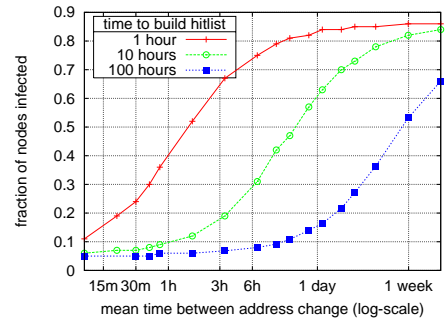


Figure 10. Maximum fraction of infected hosts vs. time between host address changes for different hitlist rates assuming scan-blocking mechanisms

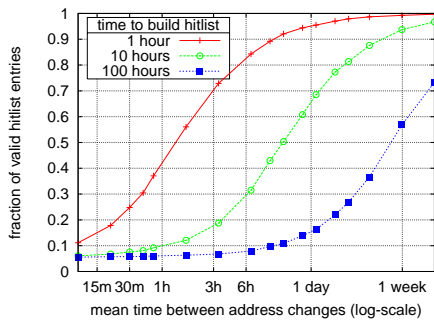


Figure 11. Effect of network address space randomization on hitlist decay

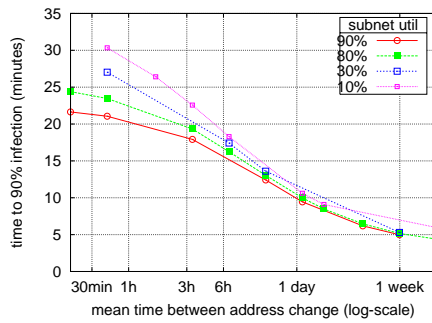


Figure 12. Effect of network address space randomization vs. subnet usage density

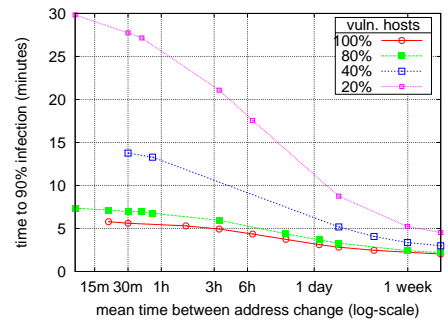


Figure 13. Effect of network address space randomization for different vulnerable host populations

spread time, expressed in terms of the time required for the worm to infect 90% of the vulnerable hosts. We compare the impact of network address space randomization, varying how fast the hitlist is generated and how fast the host addresses are changed. The fraction of vulnerable hosts at 20%, the internal scan success probability is 0.3 (based on the subnet utilization measurements of Section 4.4) and the random scanning success probability is 0.05 (based on the measurements presented in Section 4.1).

The results are shown in Figure 8. We observe that NASR achieves the goal of increasing the worm spread time, from 5 minutes when no NASR is used to between 24 and 32 minutes when hosts change their addresses very frequently. As expected, defending against hitlists that are generated very fast requires more frequent address changes; in general, the mean time between address changes needs to be 3-5 times less than the time needed to generate the hitlist for the

approach to reach around 80% of its maximum effectiveness. More frequent address changes give diminishing returns. Considering the observations of Section 4, it appears that daily address changes could significantly slow down a worm whose hitlist is generated by passively listening to a P2P network.

Note that when using NASR, the hitlist worm is not completely reduced to a random-scanning worm: knowledge of subnets that have at least one host available already gives the worm some advantage over a completely oblivious, random-scanning worm. In this experiment, such a worm would require 2 hours to infect the whole network. This is the result of performing subnet-level instead of global-level NASR, as global-level NASR would indeed reduce the hitlist worm to random-scanning. We must also note that the spread times reported depend on scanning frequency, although the relative improvement when using NASR is constant.

We also simulated NASR with varying the number of vulnerable hosts, and the average subnet utilization. The impact of NASR is greater in terms of slowing down the infection for smaller vulnerable populations. This is expected, as in such cases the failure rate for stale entries is higher compared to a network where every available host is vulnerable. The results for the impact of NASR as a function of subnet utilization are similar. Higher utilization means a higher success rate for stale entries. However, NASR remains effective even for 90% subnet utilization.

5.2 Partial deployment scenario

We have so far assumed that NASR is globally deployed. In reality, it is more likely that only a fraction of subnets will employ the mechanism, such as dynamic address pools. As we are not aware of any studies estimating the fraction of DHCP pools, we measure the effectiveness of NASR for different values for the fraction of NASR-enabled subnets. The results are shown in Figure 9. We observe that NASR continues to be effective in slowing down the worm, even when deployed in 20% or 40% of the network. The worm will still infect the non-NASR subnets quite rapidly with a slowdown in the order of 50%, caused by the worm failing to infect NASR subnets. In other words, NASR has a milder but still positive impact on non-NASR hosts. However, the worm will have to resort to random scanning after exhausting the hitlist, and it will take significantly more time to infect NASR compared to non-NASR subnets. This provides a clear incentive for administrators to deploy NASR, as it may provide them the critical amount of time needed to react to a worm outbreak.

5.3 Interaction with scan-blocking mechanisms

Hitlist worms are generally immune to scan-blocking mechanisms such as [23]. Even for the natural decay rates measured in Section 4, such worms would still be *under* the detection threshold most of the time. Randomization, however, will cause many infection attempts to fail, as hosts change addresses and their previous addresses are either unused or used by a different host that may or may not run the same service, and thus may or may not be vulnerable. To determine the interaction between NASR and scan-blocking mechanism we simulate worm outbreaks in a network where both NASR and scan-blocking are deployed. As scan-blocking *contains* the outbreak, in this experiment we measure the maximum frac-

tion of hosts that are infected despite NASR and scan-blocking. The results are shown in Figure 10. We observe that if NASR is performed according to the rule-of-thumb observation made previously (e.g., with address changes at a rate that is 3-5x faster than hitlist generation), the infection can be contained to under 15% of the vulnerable population.

6 Discussion

The experiments presented in Sections 4 and 5 suggest that network address space randomization is likely to be useful. However, these results should only be treated as preliminary, as there are several issues that need to be examined more closely before reaching any definite conclusions.

First, the interaction between NASR and other defense mechanisms needs to be studied in more depth. Our simulation results show that NASR enables scan-blocking mechanisms to contain the worm to under 15% infection. However, scan-blocking is not entirely foolproof, at least in its current form. For example, a list of *known repliers* can be used to defeat the failed-connection test used by these mechanisms, by padding infection attempts with successful probes to the known repliers. Whether it is possible to design better mechanisms for detecting and containing scanning worms is thus still an open question. Therefore, we should also consider other possibilities, including distributed detection mechanisms. As NASR is likely to at least slow down worms, it may provide the critical amount of time needed for distributed detectors such as [8, 26] to kick in. Determining whether this is indeed a possibility requires further experimentation and analysis.

Second, we have so far focused entirely on IP-level address randomization, as IP hitlists seem to be the most dangerous in the current Internet. For instance, we have only considered IPv4 as deployed today. In an IPv6 Internet, the address space is so much bigger that randomization could be even more effective. We also assume that worms that use higher-level addressing schemes, such as DNS or DHT id's, will suffer the additional lookup cost and risk of being detected. With some simplifications, we can see how this is true for the case of DNS. IP-level NASR would be rendered useless if a DNS name hitlist is used, for example, for attacking Web servers, for which the DNS name will have to be updated under IP randomization so that *www.site.com* always points to the correct IP. We measured the fully qualified domain name (FQDN) for several entries from search engine results. The average length was 16 bytes. Servers that hold web

content tend to have shorter, more memorable names, so we expect that this is a conservative estimate. We measured a 46% compression ratio for these strings, and therefore on average each entry will take up 7.5 bytes in the hitlist. IP addresses take up 4 bytes, so storing DNS names causes almost a doubling of the hitlist size. The DNS lookups required for resolving the names also introduce latency. Resolving the names used in the previous paragraph results in an average latency of 1 second. It is possible to pipeline these requests, but massive DNS lookups may raise suspicion. While no such mechanism is in place now, it could be deployed on DNS servers. Thus, while the worm is spread through the entire Internet, in practice, every infection has to be processed by the DNS system, involving orders of magnitude less hosts.

Third, we have not considered how worm creators would react to widespread deployment of our mechanism. One option would be for the worm to perform a second round of (stealthy) probing, and retain only entries that seem to be stable over time. If our mechanism is partially deployed, then the worm could infect the non-NASR part of the Internet, without being throttled by stale entries or generating too many failed connections. Interestingly, in this scenario all networks that employ NASR will be worm-free, unless the worm switches to random scanning after finishing with the hitlist. If this happens, then NASR networks will still get infected much later than the nodes in the hitlist. Although we are not aware of any other possible reactions to the deployment of our mechanism, we cannot safely dismiss the possibility that worm creators could come up with other measures to counter NASR. Thus, this issue deserves further thought and analysis.

7 Summary and concluding remarks

We have explored the design and effectiveness of *network address space randomization* (NASR), a technique that hardens IP networks against hitlist worms. The idea behind NASR is to force network nodes to periodically change their network address in order to increase the staleness of information contained in hitlists. The approach is appealing in many ways. First, it slows down hitlist worms, and forces them to exhibit scan-like behavior that may be detectable using other mechanisms, where available. Second, it is different in nature from most previously proposed worm defenses, as it is neither a detection mechanism that needs to analyze network activity, nor an end-system enhancement. This makes the approach very easy to

implement and deploy at low operational cost.

We have discussed various constraints that limit the applicability of the proposed approach, such as services that use hardwired addresses (such as routers, DNS servers, etc.) and those that cannot tolerate address changes, or suffer performance-wise when addresses change frequently. It appears that network segments that *already* perform dynamic address allocation, such as DHCP pools for broadband connections, laptop subnets and wireless networks could easily implement the mechanism without significantly impairing end host functionality.

To explore the design space of NASR we first considered some basic hitlist characteristics, such as the speed at which a hitlist can be constructed, the natural rate of hitlists without NASR, and how address space is allocated and utilized. We performed measurements on the Internet to obtain a clearer picture of these characteristics. The measurements show that hitlists built by ICMP ECHO scans are relatively robust, with the fraction of entries that are stale at around 25%, two weeks after hitlist generation. Harvesting search engines results in very low decay rates but are typically harder to build without raising suspicions. Passively snooping on a P2P network can very quickly populate a hitlist with hundreds of thousands of hosts, without generating any suspicious activity. However, P2P hitlists also decay faster.

Additional measurements show that most subnets tend to be utilized up to 50%, while 65%-95% of the subnets are less than 12% full. This observation suggests that hitlist worms will result in failed connections and timeouts when hitting a stale entry. Further measurements show that the natural uptime of hosts (e.g., without NASR) is around 5 days for between 20% and 60% of the hosts, while 20%-40% of all hosts have an uptime of less than 12 hours. Although the methodology for these measurements is conservative, the results indicate that if daily address changes are good enough to slow down or contain a worm, the cost of NASR is almost free for 20%-40% of the hosts, but may have some implications for some of the remaining 20%-60% of the hosts.

Finally, our preliminary simulation results suggest that network address space NASR slows down the spread of the worms. We show that NASR forces hitlist worms to exhibit scanning-like properties, exposing them to previously proposed scan-detection mechanisms, and thus making the worm easier to contain.

Acknowledgments

This work was supported in part by the IST project SCAMPI (IST-2001-32404) funded by the European Union and the GSRT project EAR (USA-022) funded by the Greek Secretariat for Research and Technology. The work of K. G. Anagnostakis is also supported in part by ONR under Grant N00014-01-1-0795. P. Akritidis, E. P. Markatos, and S. Antonatos are also with the University of Crete. The work of K. G. Anagnostakis was done while at ICS-FORTH.

We are indebted to the network administrators at FORTH, UoC and the anonymous ISP for agreeing to tolerate our experiments, and for providing us with detailed information on how their subnets are allocated. We also thank Sotiris Ioannidis for his “constructive” comments on an earlier draft of this paper.

References

- [1] CERT Advisory CA-2001-19: ‘Code Red’ Worm Exploiting Buffer Overflow in IIS Indexing Service DLL. <http://www.cert.org/advisories/CA-2001-19.html>, July 2001.
- [2] Cert Advisory CA-2003-04: MS-SQL Server Worm. <http://www.cert.org/advisories/CA-2003-04.html>, January 2003.
- [3] The Spread of the Sapphire/Slammer Worm. <http://www.silicondefense.com/research/worms/slammer.php>, February 2003.
- [4] DISCO: The Passive IP Discovery Tool. <http://www.altmode.com/disco/>, 2004.
- [5] Fingerprinting: The complete documentation. <http://www.10t3k.org/security/docs/fingerprinting/>, 2004.
- [6] Fingerprinting: The complete toolbox. <http://www.10t3k.org/security/tools/fingerprinting/>, 2004.
- [7] THC-Amap. <http://thc.org/releases.php>, 2004.
- [8] K. G. Anagnostakis, M. B. Greenwald, S. Ioannidis, A. D. Keromytis, and D. Li. A Cooperative Immunization System for an Untrusting Internet. In *Proceedings of the 11th IEEE International Conference on Networking (ICON)*, pages 403–408, September/October 2003.
- [9] Michael Atighetchi, Partha Pal, Franklin Webber, Rick Schantz, and Chris Jones. Adaptive use of network-centric mechanisms in cyber-defense. In *Proceedings of the 6th IEEE International Symposium on Object-oriented Real-time Distributed Computing*, May 2003.
- [10] Ricardo A. Baratto, Shaya Potter, Gong Su, and Jason Nieh. Mobidesk: mobile virtual desktop computing. In *Proceedings of the 10th Annual International Conference on Mobile Computing and Networking (MOBICOM)*, pages 1–15. ACM Press, 2004.
- [11] F. Cohen. Computer Viruses: Theory and Practice. *Computers & Security*, 6:22–35, February 1987.
- [12] J. Jung, V. Paxson, A. W. Berger, and H. Balakrishnan. Fast Portscan Detection Using Sequential Hypothesis Testing. In *Proceedings of the IEEE Symposium on Security and Privacy*, May 2004.
- [13] Gaurav S. Kc, Angelos D. Keromytis, and Vassilis Prevelakis. Countering Code-Injection Attacks With Instruction-Set Randomization. In *Proceedings of the ACM Computer and Communications Security Conference (CCS)*, pages 272–280, October 2003.
- [14] Jeffrey O. Kephart. A Biologically Inspired Immune System for Computers. In *Artificial Life IV: Proceedings of the Fourth International Workshop on the Synthesis and Simulation of Living Systems*, pages 130–139. MIT Press, 1994.
- [15] 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 2nd Internet Measurement Workshop (IMW)*, pages 273–284, November 2002.
- [16] D. Nojiri, J. Rowe, and K. Levitt. Cooperative response strategies for large scale attack mitigation. In *Proceedings of the 3rd DARPA Information Survivability Conference and Exposition (DISCEX)*, April 2003.
- [17] S. E. Schechter, J. Jung, and A. W. Berger. Fast Detection of Scanning Worm Infections. In *Proceedings of the 7th International Symposium on Recent Advances in Intrusion Detection (RAID)*, pages 59–81, October 2004.
- [18] C. Shannon and D. Moore. The spread of the witty worm, 2004. <http://www.caida.org/analysis/security/witty/>.
- [19] S. Staniford. Containment of Scanning Worms in Enterprise Networks. *Journal of Computer Security*, 2004.
- [20] S. Staniford, D. Moore, V. Paxson, and N. Weaver. The top speed of flash worms. In *Proc. ACM CCS WORM*, October 2004.
- [21] S. Staniford, V. Paxson, and N. Weaver. How to Own the Internet in Your Spare Time. In *Proceedings of the 11th USENIX Security Symposium*, pages 149–167, August 2002.
- [22] N. Weaver and V. Paxson. A worst-case worm. In *Proc. Third Annual Workshop on Economics and Information Security (WEIS’04)*, May 2004.
- [23] N. Weaver, S. Staniford, and V. Paxson. Very Fast Containment of Scanning Worms. In *Proceedings of the 13th USENIX Security Symposium*, pages 29–44, August 2004.
- [24] M. Williamson. Throttling Viruses: Restricting Propagation to Defeat Malicious Mobile Code. Technical Report HPL-2002-172, HP Laboratories Bristol, 2002.
- [25] Jian Wu, Sarma Vangala, Lixin Gao, and Kevin Kwiat. An Effective Architecture and Algorithm for Detecting Worms with Various Scan Techniques. In *Proceedings of the Network and Distributed System Security Symposium (NDSS)*, pages 143–156, February 2004.
- [26] Vinod Yegneswaran, Paul Barford, and Somesh Jha. Global Intrusion Detection in the DOMINO Overlay System. In *Proceedings of the Network and Distributed System Security Symposium (NDSS)*, February 2004.
- [27] C. C. Zou, L. Gao, W. Gong, and D. Towsley. Monitoring and Early Warning for Internet Worms. In *Proceedings of the 10th ACM International Conference on Computer and Communications Security (CCS)*, pages 190–199, October 2003.
- [28] C. C. Zou, W. Gong, and D. Towsley. Code Red Worm Propagation Modeling and Analysis. In *Proceedings of the 9th ACM Conference on Computer and Communications Security (CCS)*, pages 138–147, November 2002.