Worm Meets Beehive

Xuxian Jiang, Dongyan Xu, Shan Lei, Paul Ruth
Department of Computer Sciences
Purdue University
West Lafayette, IN 47907
{ *jiangx, dxu, leishan, ruth*}@cs.purdue.edu

Jianzhong Sun
Department of Mathematics
Purdue University
West Lafayette, IN 47907
sunj@ics.purdue.edu

***Abstract*— Internet worms continue to plague the Internet infrastructure with wider and deeper impact since the Morris Worm in early 1988. It has been further shown that better-engineered worms like *Warhol* worms and *Flash* worms could spread across the Internet in minutes or even tens of seconds rather than hours. Such virulent spreading invalidates any manual counter-measures and poses an extremely serious threat to the safety of the Internet.**

To address this challenge, this paper proposes a novel worm-curtailing scheme, i.e., *beehive*, which is able to *fight-back* worm propagation by actively immunizing any encountered worm-infected node. More specifically, by owning a portion of the unused but routable IP space that is open to infection attempts of different worms, a beehive not only attracts and traps these attempts, but also defensively gives a *security shot* to each attempting worm-infected node. The security shot will immunize the infected node so that the node will not be able to infect others. Our formal analysis shows that even one beehive network with a reasonable IP address space can effectively mitigate active spreading of worms among a million nodes. This paper presents both analysis and simulation results of beehive evaluation. Particularly, our results show that for a random-probing worm, a $/13$ beehive network or 8 class B networks are able to reduce the maximum worm infection coverage to as low as 13%. To the best of our knowledge, no such worm fight-back mechanism has been proposed and analyzed before. Finally, a beehive prototype is presented to demonstrate its practicality.

***Index Terms*—security, worm, modeling, simulation, system design**

I. Introduction

Since the Morris Worm of early 1988, the persistent existence and destructive spreading of worms have posed significant threats to the shared Internet infrastructure. Recent worm incidents like Code Red worms [1] and MS-Blaster worms [8] have just warned us how fragile our networks are and how fast a virulent worm can spread. Even worse, better-engineered worms like *Warhol* worms and *Flash* worms could spread across the Internet in minutes or even tens of seconds rather than hours [15]. Also, with current computer systems becoming more and more complicated, it is more difficult than before to eliminate software bugs. In fact, new more security vulnerabilities have been discovered on a daily basis and exploiting worms have been observed more frequently than before: Code Red worms [1] and Nimda worms [6] in 2001, SQL-Slammer worms [7] in 2002, MSBlaster worms [8] and SoBig worms [10] in 2003, and MyDoom worms [11], Witty Worms [12], and Sasser worms [13] in 2004. This serious situation poses great challenges for the effective containment of fast-spreading worms.

There are few answers, either proactive or reactive, to the worm threat. Proactive approach puts the computer systems always on alert for potential vulnerabilities and tries to seal vulnerability holes before they are exploited. For example, the Windows Update Service checks the availability of security patches and applies them to eliminate security defects [6][7][8] in a timely fashion. However, experience [27] has shown that end-users or even network administrators often do not install security patches even long after they are made available because of the following concerns [29]:

- Service disruption: Applying a patch typically involves restarting affected host service or even restarting the entire host system, which may not be acceptable for critical services.
- Patch unreliability and irreversibility: Security patches are released as quick responses for identified vulnerabilities and may not have been fully verified. Moreover, installing a security patch is a commonly irreversible operation; once a patch is applied, there is no easy way to un-install the patch.

Other worm containment schemes are also proposed: Moore *et al.* [22] proposed *Address blacklisting* and *content filtering* to isolate worms; Williamson *et al.* [14] suggested modifying the network stack to throttle the worm propagation rate; Chen *et al.* [23] described a distributed anti-worm architecture (DAW) to slow down worm prop-

agation.

This paper presents a novel and complementing worm-repressing scheme called *beehive*. Residing in an unused but routable network which would be, either randomly or preferably, attacked by different worms, a beehive not only attracts and traps these attempts, but defensively gives a *security shot* to each infected node that is attempting to infect an IP belonging to the beehive. The security shot will then inoculate the infected node. As we will show, a beehive with a reasonably large IP space can effectively mitigate active spreading of worms among a million vulnerable nodes. The contributions of this paper are threefold:

- First, a new mechanism is proposed to suppress worm propagation by actively fighting back and immunizing attempting worm instances. To the best of our knowledge, no such worm fight-back mechanism has been proposed previously.
- Second, this paper conducts a formal study of beehive and presents associated models based on two worm models, i.e., the classic *epidemic* model and the *two-factor* worm model [20]. Both numerical and simulation results show promise.
- Third, a signature-based beehive prototype has been built to demonstrate its practicality and effectiveness.

The rest of this paper is organized as follows: Section II presents background on worms and describes how beehive can be used to contain worm propagation. The following section analyzes beehive using the classic epidemic worm model and shows analytical expressions, numerical solutions, and simulation results. Furthermore, beehive is modeled based on a recently proposed *two-factor* worm model [20] to demonstrate its generality. In Section IV, beehive deployment issues are discussed. Our beehive prototype is presented in Section V. Finally, Section VI examines related work and Section VII concludes this paper.

II. The Beehive Approach

This paper focuses on worms replicating themselves without human interactions by remotely exploiting *known* vulnerabilities in operating systems or application services. If we break down the actions of these worms [1] [8] [13], the following common behaviors or stages will be exposed:

- *Target Selection* This stage picks up a target either randomly [1] or with certain local subnet preference [8]. A simple ICMP or TCP *syn* network probe could be used to locate a node running a vulnerable service. The vulnerability can be remotely exploited by this worm. Once such a node is identified, an ensuing exploitation attempt will be observed.
- *Exploitation* Successful exploitation relies on the discovery of a particular vulnerability in the victim node. It is often true that worms take advantage of *well-known* vulnerabilities and *published* exploits to compromise their victims.
- *Replication* A worm infection is not completed until a replica is successfully transferred from the infecter to the victim. However, the boundary between this stage and the exploitation stage is often blurred: some worm could contain a copy of itself as the payload during the exploitation; and others may have an explicit process downloading a worm replica. A completed replication converts the victim to a worm node, which is able to begin infecting others.

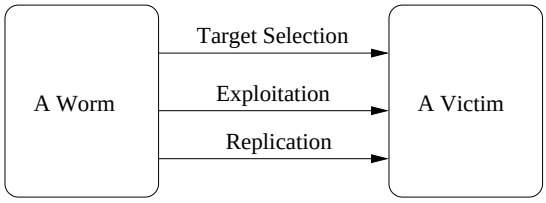


Fig. 1. Staged View of Worm Infection

Based on the staged view of worm infection, there exist initially two classes of nodes: *infectious* nodes and *vulnerable* nodes. However, once a security patch is applied to either a *vulnerable* node or an *infectious* node, the node will be turned into an *inoculated* node. Virulent worms attempt to *move* more nodes from vulnerable status to infectious status, while worm containment strategies strive to either slow down such movement or inoculate more nodes from either vulnerable state or infectious state. Unfortunately, the existence of worm outbreaks shows faster worm *infection* rates than *inoculation* rates.

However, if the victim in Figure 1 is able to fight back the attempted infection, a significant difference can be made. This paper explores such a "fight-back" scheme and proposes a special "defensive" victim - the beehive. A beehive can be thought of as an immunization service guarding an unused but routable IP subspace. An unused IP space has the advantages of collecting and monitoring highly concentrated suspicious traffic, and meanwhile avoiding possible disturbance to production networks.

The beehive has knowledge about currently *known vulnerabilities*. With the IP space "attracting" the worms, the beehive is able to monitor and identify the worms' ex-

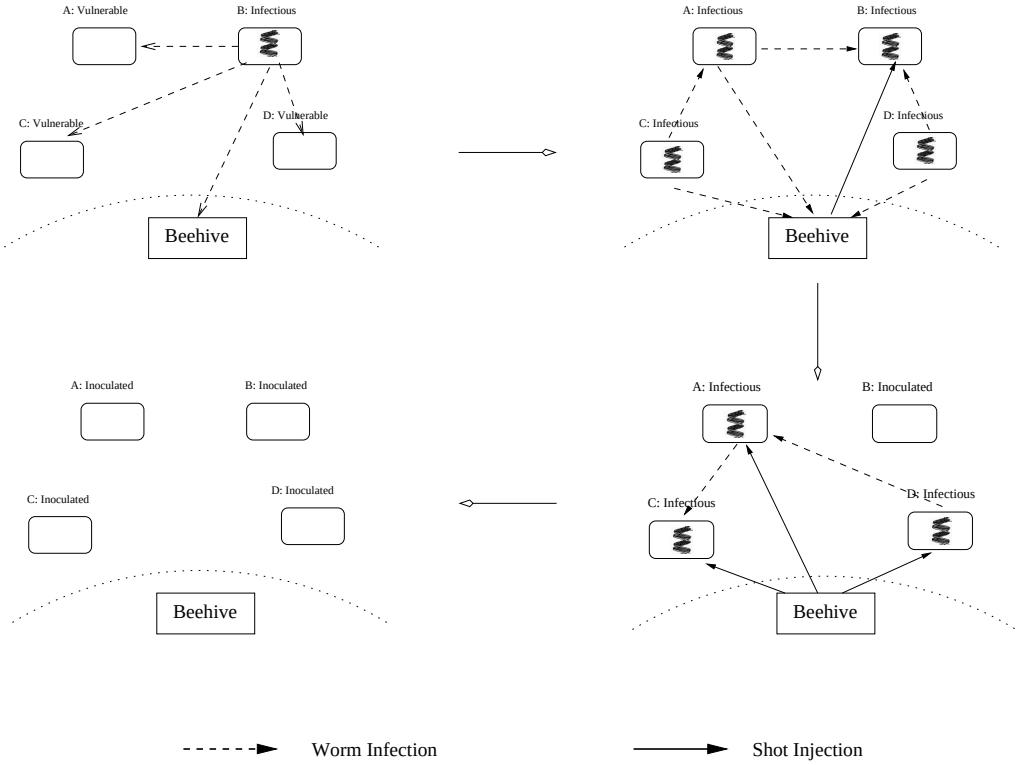


Fig. 2. Interactions Between Beehive and Worms

ploitation behavior associated with the known vulnerabilities. The beehive will then identify the infecting node, and defensively inject a *vulnerability-specific* security shot to immunize the node. We note that due to the associated risks and responsibilities, a beehive must be managed by a trusted authority.

Figure 2 shows a typical interaction between a beehive and worm instances. Residing in a routable and unused network space, a beehive will detect various worm infection attempts. For simplicity, the figure only shows the beehive and four vulnerable nodes.

1) At first, node B, the only infectious node, will spawn several threads to simultaneously probe and infect other nodes: A, C, D, and the beehive (more specifically, an IP belonging to the beehive).
2) Since nodes A, C, and D are vulnerable, the infection attempts result in successful replication of the worm from B to A, C, and D. The nodes A, C, and D will then be instructed to activate the worm and thus become infectious nodes attempting to pass the worm to others. However, when node B is infecting the beehive, the latter is able to discern such infection attempt and a security shot is injected to immunize B.
3) Nodes A, C, and D, which are now infectious, will also actively seek to infect other nodes. If the beehive's IP address space is reasonably large, it may receive some of the infection attempts. If attacked, beehive will fight-back by injecting security shots to the infectors.
4) Suppose the security shots are able to successfully immunize the infecting nodes, some (if not all) previously vulnerable nodes will become inoculated and thus the worm propagation will be mitigated or even stopped.

The scheme of beehive are based on the following two assumptions:

- *The trusted authority managing beehive should be allowed to inject security shots to those worm-infected nodes that are* **actively** *infecting others.* This assumption is justifiable from the following observations: (1) Worms infecting nodes not only cause disturbance in the infected nodes, but also generate lots of traffic affecting other normal nodes and the whole Internet. The vast detrimental impacts, including networked service disruption and business slowdown, demand an immediate stoppage of infecting nodes. (2) The defensive-only security shot, which is *signed* and administrated by a trusted authority, is a technically harmless response with the benign intention of immunizing infected nodes [29].
  Although this assumption may not be universally ac-

ceptable in the Internet, it will become more realistic and acceptable, when different administration domains, like ISPs, deploy their own beehives to protect their internal hosts and cooperate with each other in tracking down worms propagating across different domains. If a beehive detects a worm infection from its own internal domain, it can directly inject the security shot to immunize the node; if the infection is from another domain, the beehive could collect the *evidence* of the infection, and send a *signed* copy of this evidence to the beehive responsible for the other domain. The latter beehive could then take appropriate actions based on the evidence. Also within a large ISP network, a hierarchy of beehives may be deployed, each being authoritative over its own subnetwork.

```
#!/bin/sh
# launch the exploit against the internal infected attacker
# then execute commands to purify the ugly victim
/usr/local/bin/evil_exploit_dcom -d $1 -t 1 -l 4445 << EOF
taskkill /f /im msblast.exe /t
del /f %SystemRoot%\System32\msblast.exe
echo Windows Registry Editor Version 5.00 > c:\cleaner_msblast.reg
echo [HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows\CurrentVersion\Run]
>> c:\cleaner_msblast.reg
echo "windows auto update" = "REM msblast" >> c:\cleaner_msblast.reg
regedit /s c:\cleaner_msblast.reg
del /f c:\cleaner_msblast.reg
shutdown -r -f -t 0
exit

EOF
```

Fig. 3. Remotely Cleaning a MSBlaster-Infected Node

- *It is technically feasible to inject a security shot to a worm-infected node and the shot is able to protect the node from both incoming and outgoing exploitation.* Considering current network security reality and the way worms infect hosts using remotely exploitable vulnerabilities, we argue that a security shot can be injected in a way similar to worm infections [1]. Here we cite one real-world example justifying this assumption: during the MSBlaster worm outbreak in August, 2003, Oudot Laurent, a security expert, wrote a script [30] (Figure 3) which is able to *remotely* clean a MSBlaster-infected node. Once an infectious node is identified, the script is able to kill the MSBlaster process in that node and also remove affected registry entries.
  However, the script itself in Figure 3 does not prevent the node from being affected again and thus it is still not enough for beehive purpose. More importantly, it requires the worm's signature. To further immunize the node without specific worm signatures, we need a more powerful technique. Security patches are able to seal vulnerability holes independent of worm signatures, but it may introduce service disruption. Network filters can minimize such disturbance by examining incoming and outgoing traffic and drop traffic that exploits vulnerabilities. However, regular filters either coarsely block certain port number or require known worm signatures. Fortunately, in a recent worm-blocking project *shield* [29], a vulnerability-driven network filter is developed for the prevention of known vulnerability exploits *without* knowing worm signatures. By applying their filtering technique, beehive security shots can be developed once a vulnerability is identified, *before* exploiting worms come into being.

[1] In some extreme cases, worms could download security patches to prevent vulnerabilities from being exploited. In these cases, beehive is not able to remotely inoculate infected nodes and need other mechanisms to block or filter traffic from infected nodes.

In the following section, we evaluate the effectiveness of the beehive approach and attempt to answer the following questions: How effective can a beehive be? What is the reasonable size of the associated IP space?

III. Evaluation Of Beehive

In this section, we first introduce the notations used in our analysis and describe the classic worm epidemic model. We then derive a beehive model and present its analytical and numerical solutions. Simulation results are also presented to confirm beehive's effectiveness. Finally, we analyze beehive under a more realistic *two-factor* worm model.

A. Classic Epidemic Model

This paper uses $i(t)$, $v(t)$, and $r(t)$ to represent the number of infectious nodes, the number of vulnerable nodes, and the number of inoculated nodes at time t, respectively. Also we denote the total number of nodes involved in a worm outbreak as N ($N = i(t) + v(t) + r(t)$). The notations used throughout this paper are collected in table I.

Suppose the infection rate of a certain worm is a constant α, the classic epidemic worm propagation model [33] with a finite population is defined by :

$$di(t) = \alpha \times \frac{v(t)}{N} \times i(t) \times dt \tag{1}$$

where $v(t) = N - i(t)$. $\alpha \times \frac{v(t)}{N} \times dt$ represents the number of new worm nodes contributed by a single worm source within dt period and $di(t)$ is the number of new worm nodes during the time period $[t, t + dt]$ with current worm population $i(t)$.

TABLE I

NOTATION USED IN THE PAPER

Symbol	*Description*
$v(t)$	the number of vulnerable machines at time i during the spread of worm
$i(t)$	the number of infectious machines at time i during the spread of worm
$r(t)$	the number of machines which were infected but later inoculated before time t
N	the total number of machines involved in a specific worm outbreak: N=v(t)+i(t)+r(t)
$\alpha/\alpha(t)$	the infection rate of a (self-replicating) worm at time t
B_0	the size of IP address space associated with a beehive
s	the average number of machines scanned by an infected machine per unit time

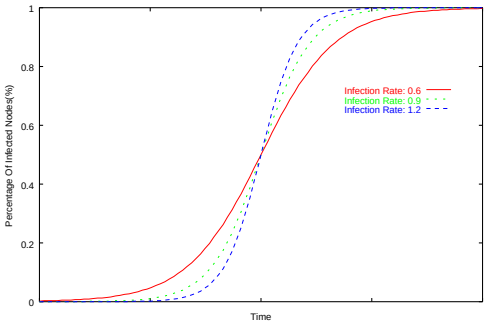


Fig. 4. Example Worm Propagation with Different Infection Rates: 0.6, 0.9, and 1.2

Eq (1) is also known as the *logistic equation* [36] and has the following solution:

$$i(t) = N - \frac{N}{1 + e^{\alpha(t-T)}} \tag{2}$$

where T is some constant dependent on the initial worm population. Based on eq (2), example worm propagations with three different infection rates are drawn in figure 4. The curves are known as the logistic curves [36] and exhibit the "sigmoid" shape.

B. Beehive Under Classic Epidemic Model

Let s be the average scanning rate of one worm source probing for victims, then during the time period $[t, t+dt]$, the number of scan attempts from one worm source is $s \times dt$, and there are $s \times dt \times i(t)$ scans in total for all $i(t)$ worm sources. Assuming the scans are uniformly distributed over all IPv4 address space (2^{32}), the probability of a machine being scanned is $\alpha(t) = 1-(1-\frac{1}{2^{32}})^{si(t)dt} \approx C_0 i(t)dt$ [2], where the constant $C_0 = \frac{s}{2^{32}}$.

With the total number of current scan attempts, the expected number of vulnerable machines that will be subverted into infectious nodes during $[t, t+dt]$ is $\alpha(t) \times v(t)$. In other words,

$$dv(t) = -C_0 i(t) v(t) dt \tag{3}$$

The minus sign shows the decreasing number of vulnerable nodes and thus the increasing number of infectious nodes due to current infection attempts.

Similarly, when a beehive guarding an unused IP address space of B_0 is deployed, the probability of a worm source hitting this IP space during time period $[t, t+dt]$ is $\beta = 1 - (1 - \frac{B_0}{2^{32}})^{sdt} \approx C_1 dt$, where the constant $C_1 = \frac{sB_0}{2^{32}} = B_0 C_0$. Therefore, the number of infectious machines that are inoculated during the time period $[t, t+dt]$ is:

$$dr(t) = \beta \times i(t) = C_1 i(t) dt \tag{4}$$

By $N = v(t) + i(t) + r(t)$, eq (4) can be rewritten as:

$$di(t) = -dv(t) - dr(t) = (C_0 v(t) - C_1) i(t) dt \tag{5}$$

Summing eq (3) and eq (5), we achieve beehive model based on the classic epidemic worm propagation:

$$\frac{d}{dt}\begin{bmatrix} i \\ v \end{bmatrix} = \begin{bmatrix} -C_1 & C_0 i \\ 0 & -C_0 i \end{bmatrix} \begin{bmatrix} i \\ v \end{bmatrix}, \quad i(0) = i_0, \quad v(0) = v_0 \tag{6}$$

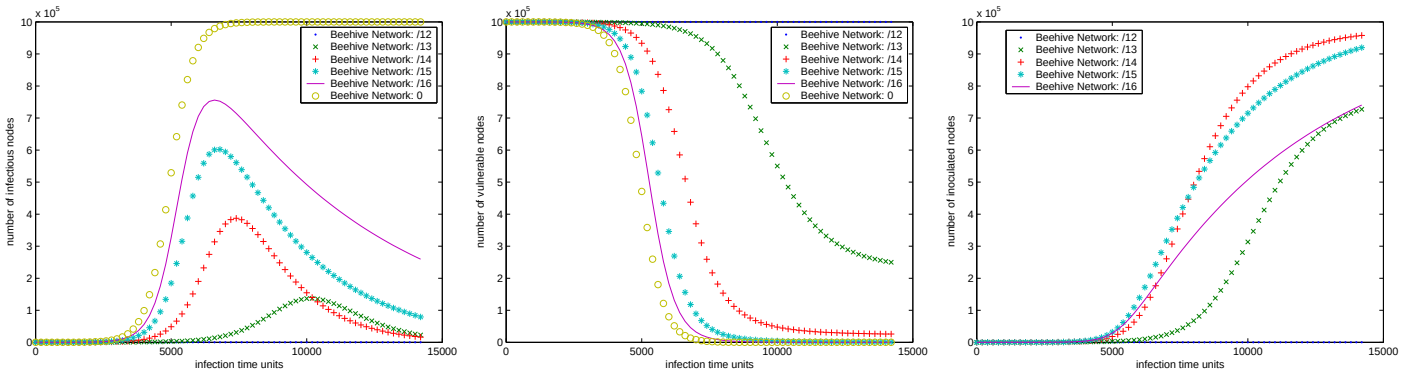
where i_0 is the initial infected nodes or "*hit-list*" [15] size, and v_0 is the initial number of vulnerable nodes.

C. Numerical and Analytical Results

Numerical solutions of eq (6) are presented in Figure 5 with varying beehive network (i.e. size of associated IP address space) from $/16$ to $/12$ [3], where the initial values

[2]The approximation is achieved by Taylor expansion based on the fact that $si(t)dt$ is much smaller than 2^{32}.

[3]The number $/16$ is the network mask length, which corresponds to a class B network with netmask $255.255.0.0$. Similarly, the number $/12$ corresponds to a network with netmask $255.240.0.0$.



(a) Number of Infectious Nodes

(b) Number of (Remaining) Vulnerable Nodes

(c) Number of Inoculated Nodes

Fig. 5. The Effectiveness of Beehive Based on Classic Epidemic Worm Model

and average scanning rate are:

$$i_0 = 10, v_0 = 10^6, s = 10$$

The value of 10^6 is a reasonable estimate for initially vulnerable nodes. For example, it is estimated that the number of nodes vulnerable to MSBlaster is $360,000$ [8]. The X-axis is in *infection time units*: each time unit is the duration of one successful worm infection session (usually several seconds to tens of seconds). The cases of different beehives are compared with the classic epidemic model with no beehive ("Beehive Network 0" case in Figure 5) and random scanning strategy is assumed. Figure 5(a), 5(b), and 5(c) show the number of infectious nodes, vulnerable nodes, and inoculated nodes (immunized by beehive), respectively. It is obvious that both the worm coverage and intensity have been further mitigated with larger beehive network. A $/13$ beehive network is able to effectively mitigate the worm propagation by reducing the maximum number of infection nodes to as low as 13% of potential maximum vulnerable nodes, while a /12 network is able to totally prevent the worm outbreak. Specially, Figure 5 exposes two interesting results:

- Figure 5(a) shows that after the 10000th time unit, the worm outbreak under a smaller beehive network ($/14$) has fewer infectious nodes (active worm instances) than the worm outbreak under a larger beehive network ($/13$).
- Figure 5(c) shows that a smaller beehive network ($/14$) inoculates more worm instances than a larger beehive network ($/13$).

These seemingly unexpected results can be explained as follows: Based on Figure 5(b), during the time period when the first result above comes up, there exist a larger number of remaining vulnerable nodes (which are *untouched* during the outbreak) in the larger beehive case than in the smaller beehive case. The reason for this is that the outbreak is put under *control* faster by a larger beehive network and thus fewer nodes are infected. The second result above can be explained using the same argument: A larger beehive is able to contain the worm outbreak earlier - more specifically - during its *slow start* phase [19].

The effectiveness of beehive can be better characterized by the *acuteness* (Λ) and *maximum coverage* (Φ) of a worm outbreak. The acuteness of an outbreak is defined as the first-order differentiation of i(t), i.e., $\Lambda = \frac{di(t)}{dt}$, while the maximum worm coverage (Φ) can be modeled as $max\{\frac{i(t)}{N} : t \geq 0\}$.

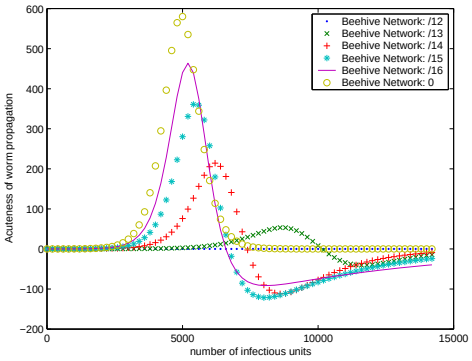


Fig. 6. The Acuteness of Worm Outbreak

Figure 6 shows the acuteness of worm outbreaks using different beehive networks. With a larger IP address space, beehive will be more capable of slowing down worm propagation. Figure 7 shows the effectiveness of beehive in reducing the maximum worm coverage. The larger the associated IP space, the more powerful the beehive in reducing the worm coverage.

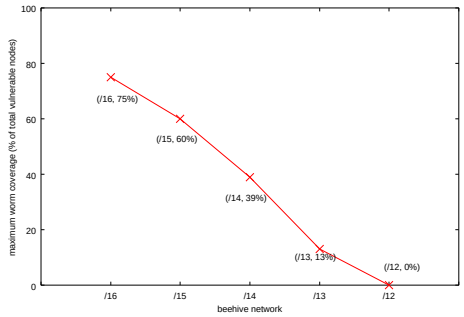


Fig. 7. The Maximum Worm Coverage

To complete our analysis, we further obtain the analytical form of maximum worm coverage Φ and the corresponding time instance.

We add up the two equations in (6) and derive the following equation:

$$\frac{d(i+v)}{dt} = -C_1 i$$

Based on eq (3), the previous equation can be re-written as:

$$\frac{d(i+v)}{dt} = \frac{B_0}{v}\frac{dv}{dt}$$

Integrating on both sides, we have:

$$i(t) + v(t) = i_0 + v_0 + B_0 \ln \frac{v(t)}{v_0} = N + B_0 \ln \frac{v(t)}{v_0}$$

Hence

$$i(t) = N - v(t) + B_0 \ln \frac{v(t)}{v_0} \tag{7}$$

When the worm propagation reaches its maximum coverage, $\frac{di}{dt}$ reaches 0. From the equation of $\frac{di(t)}{dt}$ in (6), we have

$$C_1 i(t) = C_0 i v(t)$$

Thus,

$$v(t) = \frac{C_1}{C_0} = B_0.$$

Finally, we obtain

$$\Phi = \frac{i(t)}{N} = 1 - \frac{B_0}{N} + \frac{B_0}{N} \ln \frac{B_0}{v_0}$$

Theorem 1: Assume that $(i(t), v(t))$ is the unique solution of eq (6), then i reaches its maximum value $\Phi \times N$ if and only if $v(t) = B_0$, where $\Phi = 1 - \frac{B_0}{N} + \frac{B_0}{N} \ln \frac{B_0}{v_0}$

If we further substitute $i(t)$ in eq (7) to eq (3) , we obtain

$$\frac{dv}{v(C_0 v - C_1 \ln v - C_0 C_2)} = dt \tag{8}$$

where $C_2 = N - B_0 \ln v_0$.

According to Theorem 1, $i(t)$ reaches its maximum value at $v = B_0$. If we define the corresponding time point as Γ and integrate the left-hand side of eq (8) from $v(0) = v_0$ to B_0, the result is the same as integrating the right-hand side from 0 to Γ:

$$\Gamma = \int_{v_0}^{B_0} \frac{dv}{v(C_0 v - C_1 \ln v - C_0 C_2)}$$

D. Simulation Results

To verify our numerical and analytical results, we have developed a beehive simulator based on a uniform-scan worm simulator developed by Zou [40] to confirm the effectiveness of beehive.

In the simulation, the propagation of uniform-scan worms is modeled in discrete time and the simulated system consists of N ($N = 10^6$) hosts that can reach each other directly. A host could stay in one of three states at any time: *infectious*, *vulnerable*, or *inoculated*. However, a host is in the *inoculated* state only when it is attempting to infect beehive and is thus immunized. The other simulation parameters are: $s = 10, i_0 = 10$.

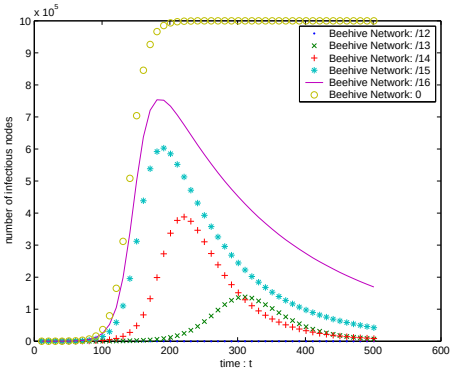


Fig. 8. Simulation: Number of Infectious Nodes

We run the simulation 100 times in a Dell PowerEdge server with a 2.6GHz Intel Xeon processor and 2GB RAM. Figure 8 shows the average number of infectious nodes with varying beehive network from /16 to /12. The time unit of X-axis is *minute*, which is used in this discrete-time-based simulator and thus is different from the infection time unit used in the numerical results. The simulation results match well with the numerical results in Figure 5(a). We further calculate two envelop curves for these 100 runs based on the maximum and minimum

values for the number of infectious hosts at each time t and find that the maximum difference between these two curves is only 0.4% to the population size N (10^6).

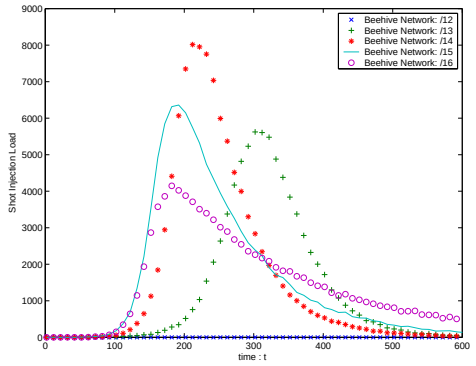


Fig. 9. Simulation: Load of Beehive

Figure 9 shows the number of security shots injected per *minute* during the worm propagation, which is interesting for understanding the possible load on beehive. The maximum number is 8200 per minute or 137 per second in the case of a $/14$ beehive network. Interestingly, the beehive workload is *not* in a linear relation with the beehive IP space size. In the Figure, the ranking of beehive peak load is $/12 < /16 < /13 < /15 < /14$. The reason is similar to that for the results in Section III-C: A beehive with a larger IP space leads to a higher worm *hit rate* and therefore quenches the worm outbreak earlier, resulting in lower worm immunization workload.

Additionally, we examine the impact of immunization time (i.e., the time to immunize one node) on the effectiveness of beehive. Figure 10 shows worm propagation under a $/14$ beehive network with varying immunization time. As expected, longer immunization time makes beehive less efficient in suppressing worm propagation, but the impact is not significant.

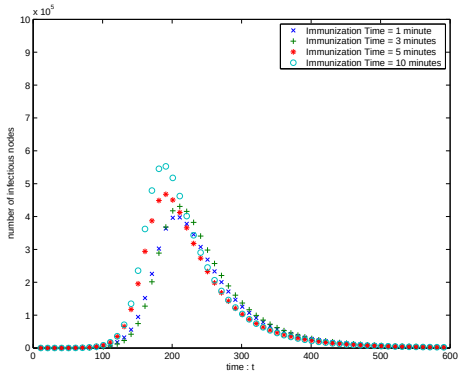


Fig. 10. Simulation: Effect of a $/14$ Beehive Network with Varying Immunization Time

E. Beehive Under Two-Factor Worm Model

The classic epidemic model is simple and does not consider other factors like network congestions and human counter-measures which could also affect the worm propagation. The *two-factor* worm model, shown in eq (9), is proposed to capture these factors and exhibits more realistic results. This subsection applies beehive to the *two-factor* model. However, beehive can be generally applied to any other worm models such as the AAWP model [25].

$$\frac{di(t)}{dt} = \alpha(t) \times [N - r_z(t) - i(t) - q_z(t)] \times i(t) - \frac{dr_z(t)}{dt} \tag{9}$$

In the *two-factor* worm model equation, $q_z(t)$ and $r_z(t)$ accommodate the effect of human counter-measures during worm propagation: $q_z(t)$ represents the number of nodes converted from the *vulnerable* state to the *inoculated* state, while the $r_z(t)$ is the number of nodes converted from the *infectious* state to the *inoculated* state. The infection rate $\alpha(t)$, which is now a variable instead of a constant, considers the impact of network congestions caused by worm propagation itself. Other notations are consistent with those in table I. This *two-factor* worm model enhances the classic epidemic model particularly during the last stage of worm propagation, i.e., the *slow finish* stage [19].

From [20], $r_z(t)$, $q_z(t)$, and $\alpha(t)$ can be represented as follows:

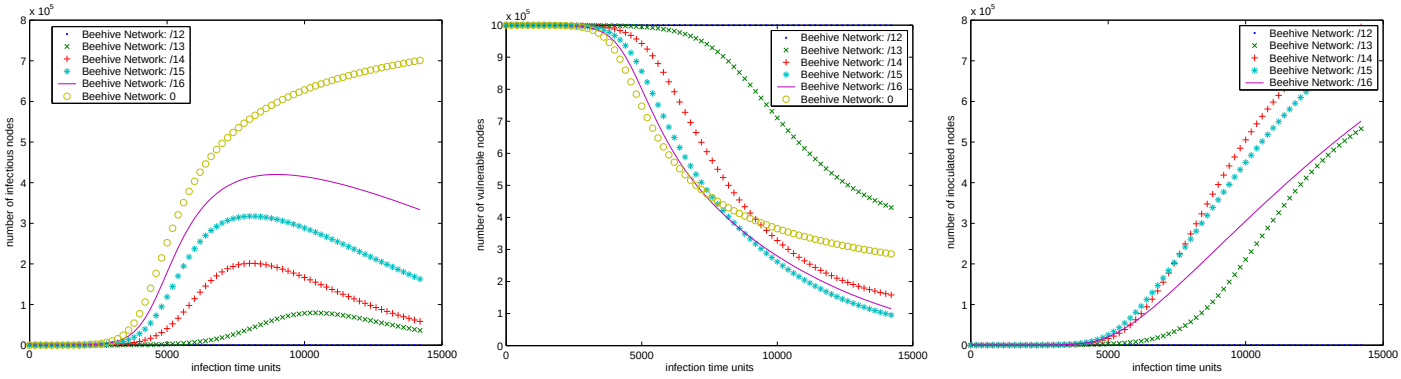
$$\begin{array}{rcl} \frac{dr_z(t)}{dt} & = & wi(t) \\ \frac{dq_z(t)}{dt} & = & uv(t)i(t) \\ \alpha(t) & = & s(1 - i(t)/N)^n \end{array}$$

where constants w and u are used to adjust the rate of converting nodes into inoculated state, and the constant n is used to adjust the infection rate so that it will be sensitive to the number of infectious hosts. When $w = 0, u = 0, n = 0$, the two factors worm model falls back to the classic model.

Finally, we derive beehive model based on eq (9):

$$\begin{array}{rcl} di(t) & = & \alpha(t)[N - r(t) - i(t)]i(t)dt - dr_z(t) \\ & & -C_1 i(t)dt \\ dv(t) & = & -\alpha(t)v(t)i(t)d(t) - dq_z(t) \\ dr(t) & = & C_1 i(t)dt + dr_z(t) + dq_z(t) \\ N & = & v(t) + i(t) + r(t) \end{array} \tag{10}$$

The term $C_1 i(t)dt$ captures the worm-immunization effect of beehive; and the r(t) here combines beehive and human counter-measures in two-factor model together and is



(a) Number of Infectious Nodes

(b) Number of (Remaining) Vulnerable Nodes

(c) Number of Inoculated Nodes

Fig. 11. The Effectiveness of Beehive Based on *Two-Factor* Worm Model

more general than the notation of r(t) in Table I.

With the following parameters: $n = 3, w = 1.5e-6, u = 2e-12, i(0) = i_0 = 10, v(0) = v_0 = 10^6, s = 10,$ we derive numerical solutions in Figures 11. The acuteness of worm outbreak is shown in Figure 12. These results confirm the generality and effectiveness of beehive approach.

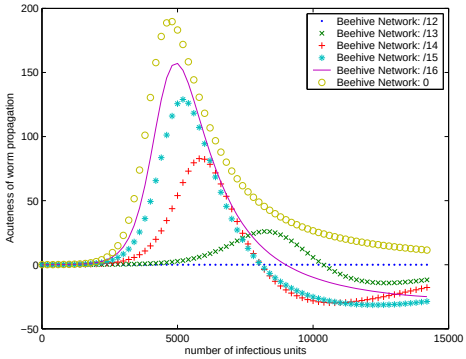


Fig. 12. The Acuteness of Worm Outbreak Under Two-Factor Worm Model

IV. Discussions

Unused IP space The previous section has shown the effectiveness of beehive in suppressing worm propagation. From the beehive models, the effectiveness largely depends on the probability of worm instances hitting the beehive. Assuming random-probing strategy during worm propagation, a high probability of hitting the beehive requires a large unused IP address space. Fortunately, beehive's IP space size requirement (e.g., a $/13$ network) seems reasonable and affordable. For example, CAIDA [1] has used a $/8$ network at UCSD and two $/16$ networks at Lawrence Berkeley Laboratory (LBL) to collect real data measuring the spread of the Code Red v2 worm. Four class B networks (a $/14$ network) have also been used as Internet sinks [24] for network abuse monitoring.

Avoiding beehive The fight-back nature of beehive may disclose its associated IP space. It is possible for a worm developer to sequentially or selectively pre-scan the Internet space for hints to locate beehive IP addresses. Once the fight-back activity is detected, the worm developer could presumably identify beehive and hard-code the corresponding network space into worm code so that it can avoid beehive.

There are several solutions: one way is to propose a *roaming* beehive whose network space is not fixed. However, this roaming approach may require more IP address space for beehive purpose. A more interesting and effective approach is to create a beehive with *scattered* rather than continuous IP addresses, which will be helpful in fighting smart topology-aware worms: each network domain *"donates"* unused IP addresses and *re-directs* traffic towards these addresses to a few beehive sites. Several recent works, including Collapsar [17] and honeyd [18], have developed techniques for the re-direction of traffic towards non-existent hosts in different domains to a centralized facility, run by a trusted authority. Furthermore, with the emergence of *sink hole networks* [31], traffic redirection overhead will be significantly reduced. All these techniques will make a beehive with scattered IP space feasible.

Fooling beehive Spoofing is a potential way to attack beehive: one worm could initiate an infection with spoofed source address. When such infection is detected by beehive, the ignorant beehive may inject a security shot

to the spoofed node. Such abusing attempts need to be detected and avoided, although the shot is not expected to do any harm to the spoofed node. Several schemes have been proposed to detect and prevent such spoofing attacks, like router-assisted source address checking and various authenticated methods to ensure the identity of communicating peers. However, extensive study on the problem is beyond the scope of this paper.

Proactive beehive The beehive presented in this paper is *reactive*. However, a *proactive* beehive can also be developed, when a vulnerability is identified and the exploiting worms have *not* yet emerged. Such *proactive* beehive can be safely deployed in each network domain, and it will actively probe and detect machines with this vulnerability *within* its own domain. Once a vulnerable machine is found, a security shot can be injected to prevent it from being exploited in the future. In this case, the first assumption in Section II will *not* be necessary.

V. A Beehive Prototype

In this section, we present a signature-based prototype to demonstrate the feasibility of beehive approach.

Figure 13 shows a generic components of a beehive: a *sensor* component and a *shot injector* component. The sensor component would either passively wait for worm probings or actively monitor real-time traffic to identify vulnerability-specific exploitations. Once an exploit is identified, the gleaned informations such as the IP address of the worm source and the vulnerabilities exploited by the worm will be given to the shot injector, which then injects an associated shot to the worm source. In the following example, we describe one beehive prototype against the Linux Adore worm [9]. The system is implemented using RedHat 7.0 Linux operating systems.

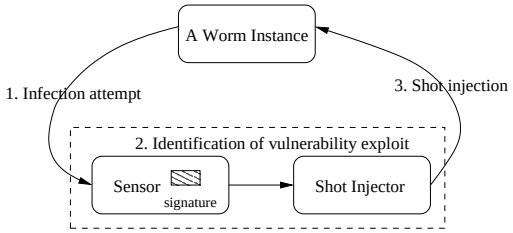


Fig. 13. Generic Components of a Beehive

The Linux Adore worm attempts to propagate itself via exploiting different un-patched services in the default installation of Linux RedHat 6.2 and 7.0 operating systems: *rpc.statd* [2], *wuftpd*[4], *BIND*[3], and *LPRng* [5]. If there is a node running any listed vulnerable service and it is successfully probed, the Adore worm will try to exploit it and execute the shell commands (shown in Figure 14) automatically regardless of which vulnerable service is exploited.

```
TERM="linux"
export PATH="/sbin:/usr/sbin:/bin:/usr/bin:/usr/local/bin"
lynx -dump http://xx.xxx.xxx/~xxxxx/red.tar >/usr/lib/red.tar
[ -f /usr/lib/red.tar ] || exit 0
cd /usr/lib;tar -xvf red.tar;rm -rf red.tar;cd lib;./start.sh
```

Fig. 14. Shell Commands Propagating The Linux Adore Worm

In the current prototype, beehive leverages an open-source IDS system, i.e., *snort* [34], as the sensor component. The particular rules detecting incoming Adore worm infection attempts are listed in Figure 15:

```
alert TCP $EXTERNAL any -> $INTERNAL any (msg: "IDS442/rpc-statdx";
flags: AP; content: "/bin|c74604|/sh";)

alert TCP $EXTERNAL any -> $INTERNAL 21 (msg: "IDS458/ftp-wuftp260";
flags: AP; content: "|31C0 31DB 31C9 B046 CD80 31C0 31DB 43 89D941 B03F CD80|";)

alert UDP $EXTERNAL any -> $INTERNAL 53 (msg: "IDS482/dns_named-exploit-infoleak";
content: "|AB CD 09 80 00 00 00 01 00 00 00 00 00 00 01 00 01 20 20 20 20 02 61|";)
alert UDP $EXTERNAL any -> $INTERNAL 53 (msg: "IDS489/dns_named-exploit-tsig";
 content: "|3F 909090 EB3B 31DB 5F 83EF7C 8D7710 897704 8D4F20|";)

alert TCP $EXTERNAL any -> $INTERNAL 515 (msg: "IDS457/LPRng-redhat7";
flags: AP; content: "|58 58 58 58 25 2E 31 37 32 75 25 33 30 30 24 6E|"; nocase;)
```

Fig. 15. Snort Rule Sets For Detecting The Infection of Adore Worms

Once the Adore sensor reports the exploitation attempt, it notifies the shot injector component with the IP address of intruding node. The injector will exploit the same vulnerabilities used by the Adore worm and inject *adore-bee.o*, a loadable kernel module containing a set of adore worm signatures. The signatures are shown in Figure 16. Though the same signatures are used, *adore-bee.o* module is able to drop both incoming and outgoing exploitation traffic matching the signatures, while the rule sets in Figure 15 are only used to detect incoming attempts.

```
drop TCP any any -> any any (msg: "IDS442/rpc-statdx";
flags: AP; content: "/bin|c74604|/sh";)

drop TCP any any -> any 21 (msg: "IDS458/ftp-wuftp260";
flags: AP; content: "|31C0 31DB 31C9 B046 CD80 31C0 31DB 43 89D941 B03F CD80|";)

drop UDP any any -> any 53 (msg: "IDS482/dns_named-exploit-infoleak";
content: "|AB CD 09 80 00 00 00 01 00 00 00 00 00 00 01 00 01 20 20 20 20 02 61|";)
drop UDP any any -> any 53 (msg: "IDS489/dns_named-exploit-tsig";
 content: "|3F 909090 EB3B 31DB 5F 83EF7C 8D7710 897704 8D4F20|";)

drop TCP any any -> any 515 (msg: "IDS457/LPRng-redhat7";
flags: AP; content: "|58 58 58 58 25 2E 31 37 32 75 25 33 30 30 24 6E|"; nocase;)
```

Fig. 16. Signatures in *adore-bee.o* Module

Once planted, the *adore-bee.o* module will be inserted into the kernel and scrutinize every incoming and outgoing packets for potential vulnerability-exploiting traffic. Once a packet matches a signature in the signature set, the packet is marked as worm-related traffic and thus dropped. The performance overhead is marginal [16] since there are only one or two signatures related to a worm in general. The reason why the Linux Adore worm requires five signatures is that it is a multi-vector worm and has the ability to propagate itself through various channels.

Due to the associated risks and lack of a scale worm testbed, the system has just been deployed in a specifically configured local network with two nodes. However, it has shown both practicality and effectiveness by successfully injecting a security shot from one node running beehive to immunize another node running the Linux Adore worm. To fully validate beehive, a scale worm testbed with tens of thousands of nodes is needed. The DETER [38] project is proposed to address this need, and beehive experiments can be performed on top of it once it is available. In the meantime, we plan to extend our beehive prototype to address the following issues:

Heterogeneity Different system platforms and different operating systems impose different requirements in designing and implementing security shots. Though recent active worms will usually propagate on one particular platform or one type of operating system, there are certain worms which are able to propagate on multiple platforms. Developing techniques to cope with heterogeneity is a challenging problem.

Signature independence We note that the need for worm signatures in our current beehive prototype is only for implementation convenience. The signature-independent technique proposed in *shield* [29] will be applied to beehive security shot concoction. As soon as a vulnerability is identified, a shield-style security shot will be installed in the beehive to fight back any exploiting worm in the future.

VI. Related Work

Proactive or *manually reactive* approaches are among the most common practices in preventing or mitigating spreading of worms. As shown in section I, although *proactive* approaches like Windows Update Service are effective, it has shown reluctance in acceptance due to the concern of unreliability and potential service disruption. Also, any manual counter-measures will be invalidated by virulent spreading of worms.

From section II, effective worm containment mechanisms should be able to either *decrease* the flow from the *vulnerable* state to the *infectious* state, termed as flow IN, or *increase* the flow from the *infectious* node to the *inoculated* state, represented as flow OUT, or the combination. The flow from the *vulnerable* state to the *inoculated* state is able to decrease the number of vulnerable nodes and thus indirectly reduce the flow IN. We classify related work based on their impact on these two flows:

Decreasing the flow IN Most of current counter-measures fall in this category. Moore *et al.* [22] explores the effect of dynamic quarantine in confining worm propagation. In particular, two defense strategies are examined: *blacklisting known infected nodes* and *filtering connections exhibiting worm signatures*. However, they require the existence of an efficient event notification system for the awareness of detected infected nodes and worm signatures. Williamson *et al.* [14] suggests modifying the network stack so that the spreading rate of worms could be slowed down. Such an approach requires the modification of commodity operating systems to be effective. Chen *et al.* [23] proposes two rate-limiting algorithms, either *temporal* or *spatial*, to mitigate the worm propagation at the ISP level. The algorithms are based on the behavioral difference between normal hosts and worm-infected hosts. Particularly, a worm-infected host has a much higher connection-failure rate when it scans the Internet with randomly selected addresses. LaBrea [35] is a tool that is able to create virtual presences on behalf of those unused IP addresses on a network. The virtual presences is able to reply the probing attempts in such a way that makes the worm instances "stuck". Beehive approach is different from, and complement, these approaches, since it can reduce the number of infectious nodes by *actively immunizing* them.

Increasing the flow OUT This category requires efficient worm *identification* and *isolation/immunization*. Recent work [28][22] mostly focuses on the issues of worm identification and isolation, and has not addressed the immunization.

Darknet [4] is another interesting research topic. Monitoring and characterizing *background traffic* for darknets have shown promise in understanding network abuse [31] [24], sensing Internet motions [37], and inferring certain remote network events [26]. There exists little or no legitimate traffic in the darknet and any anomaly will not be obscured by voluminous production traffic. Additional *active responders* can be further deployed to discriminate between different types of activities, including intrusion or attack attempts. However, they have not addressed how to contain or suppress worms using these darknets. In another interesting direction, *honeypot* [39] has been proposed as an effective way to capture worms in the wild. Furthermore, honeypot has also been leveraged for early worm detection [19], global worm detention [17], or automatic worm signature extraction [32]. The concept of *evil honeypot* [30], which parallels beehive effort, is proposed to poison or *bite* back aggressive worms. However, no formal model and analysis have been seen so far.

[4] A darknet is a portion of routable IP space in which no active services or servers reside.

VII. Conclusion

Destructive spreading of worms exposes the fragility of current Internet infrastructure and invalidates any manual counter-measures. This paper proposes *beehive* to suppress worm propagation by directly fighting back and immunizing worm-infecting hosts. Beehive leverages the unused and routable IP space for worm infection capture; and beehive security shots are capable of shielding known vulnerabilities. The effectiveness of beehive has been evaluated and demonstrated with analysis and simulation results. Furthermore, a beehive prototype has been developed to demonstrate its feasibility.

References

[1] Code Red Worms, *CAIDA Analysis of Code-Red*, http://www.caida.org/analysis/security/code-red/

[2] Rpc.statd Vulnerability *CERT Advisory CA-2000-17 Input Validation Problem in rpc.statd* http://www.cert.org/advisories/CA-2000-17.html

[3] BIND 8.2.x Vulnerability *CERT Advisory CA-2001-02 Multiple Vulnerabilities in BIND* http://www.cert.org/advisories/CA-2001-02.html

[4] Wu-ftpd 2.6.0 Vulnerability *CERT Advisory CA-2000-13 Two Input Validation Problems In FTPD* http://www.cert.org/advisories/CA-2000-13.html

[5] LPRng Vulnerability *CERT Advisory CA-2000-22 Input Validation Problems in LPRng*, http://www.cert.org/advisories/CA-2000-22.html

[6] Nimda Worms, *CERT Advisory CA-2001-26 Nimda Worm*, http://www.cert.org/advisories/CA-2001-26.html

[7] Slammer Worms, *CERT Advisory CA-2003-04 MS-SQL Server Worm*, http://www.cert.org/advisories/CA-2003-04.html

[8] Blaster Worms, *CERT Advisory CA-2003-20 W32/Blaster worm*, http://www.cert.org/advisories/CA-2003-20.html

[9] Linux Adore Worms http://securityresponse.symantec.com/avcenter /venc/data/linux.adore.worm.html

[10] SoBig Worms, http://www.cert.org/incident_notes/IN-2003-03.html

[11] MyDoom Worms, http://us.mcafee.com/virusInfo/default.asp?id=mydoom

[12] Witty Worms, http://securityresponse.symantec.com/avcenter /venc/data/w32.witty.worm.html

[13] Sasser Worms, *What You Should Know About the Sasser Worm and Its Variants*, http://www.microsoft.com/security/incident/sasser.asp

[14] Jamie Twycross and Matthew M. Williamson *Implementing and Testing a Virus Throttle* , Proc. of the 12th USENIX Security Symposium (Security '03), San Francisco, CA Aug. 2003

[15] Stuart Staniford, Vern Paxson, Nicholas Weaver *How to 0wn the Internet in Your Spare Time* , Proc. of the 11th USENIX Security Symposium (Security '02), San Francisco, CA Aug. 2002

[16] Xuxian Jiang, Dongyan Xu, and Rudolf Eigenmann *Protection Mechanisms for Application Service Hosting Platforms*, Proc. of IEEE/ACM Int'l Symposium on Cluster Computing and the Grid (CCGrid 2004), Chicago, IL, Apr. 2004

[17] Xuxian Jiang, and Dongyan Xu *Collapsar: A VM-Based Architecture for Network Attack Detention Center*, Proc. of the 13th USENIX Security Symposium (Security '04), San Diego, CA Aug. 2004

[18] Niels Provos *A Virtual Honeypot Framework*, Proc. of the 13th USENIX Security Symposium (Security '04), San Diego, CA Aug. 2004

[19] Cliff C. Zou, Weibo Gong, Don Towsley, and Lixin Gao *Monitoring and Early Detection for Internet Worms*, Proc. of the 10th ACM Conference on Computer and Communication Security (CCS'03), Washington DC 2003

[20] Cliff C. Zou, Weibo Gong, Don Towsley, *Code Red Worm Propagation Modeling and Analysis*, Proc. of the 9th ACM Conference on Computer and Communication Security (CCS'02), Washington DC Nov. 2002

[21] David Moore, Colleen Shannon, and K Claffy, *Code-Red: a case study on the spread and victims of an Internet worm*, ACM Internet Measurement Workshop, Nov. 2002

[22] David Moore, Colleen Shannon, Geoffrey M. Voelker, and Stefan Savage, *Internet Quarantine: Requirements for Containing Self-Propagating Code*, INFOCOM 2003, San Francisco, CA Mar. 2003

[23] Shigang Chen, Yong Tang, *Slowing Down Internet Worms*, Proc. of 24th International Conference on Distributed Computing Systems (ICDCS'04), Tokyo, Japan, Mar. 2004

[24] Vinod Yegneswaran, Paul Barford, and Dave Plonka *On the Design and Use of Internet Sinks for Network Abuse Monitoring*, Proc. of 7th International Symposium on Recent Advances in Intrusion Detection Sep. 2004

[25] Zesheng Chen, Lixin Gao, and Kevin Kwiat, *Modeling the Spread of Active Worms*, INFOCOM 2003, San Francisco, CA, Mar. 2003

[26] David Moore, *Network Telescopes: Observing Small or Distant Security Events*, Proc. of the 11th USENIX Security Symposium (Security '02), San Francisco, CA, Aug. 2002

[27] Eric Rescorla, *Security Holes . . . Who Cares?*, Proc. of the 12th USENIX Security Symposium (Security '03), San Francisco, CA, Aug. 2003

[28] Nicholas Weaver, Stuart Staniford, and Vern Paxson *Very Fast Containment of Scanning Worms*, Proc. of the 13th USENIX Security Symposium (Security '04), San Francisco, CA, Aug. 2003

[29] Helen J. Wang, Chuanxiong Guo, Daniel R. Simon, Alf Zugenmaier, *Shield: Vulnerability-Driven Network Filters for Preventing Known Vulnerability Exploits*, SIGCOMM 2004, Sep. 2004

[30] Laurent Oudot *Towards Evil Honeypots ?! When they bite back* 5th Annual CanSecWest/core04 Conference, Vancouver Apr. 2004

[31] Danny McPherson, and Paul Quinn *Sink Hole Networks* 5th Annual CanSecWest/core04 Conference, Vancouver Apr. 2004

[32] Christian Kreibich and Jon Crowcroft, *Honeycomb - Creating Intrusion Detection Signatures Using Honeypots*, Second Workshop on Hot Topics in Networks, Cambridge, MA USA Nov. 2003

[33] H. W. Hethcote, *The Mathematics of Infectious Diseases*, SIAM Review, vol. 42, no. 4, pp. 599-653, 2000

[34] *Snort*, http://www.snort.org

[35] T. Liston, *LaBrea*, http://www.hackbusters.net/LaBrea/

[36] Eric W. Weisstein, *Logistic Equation*, http://mathworld.wolfram.com/LogisticEquation.html

[37] *Internet Motion Sensor*, http://ims.eecs.umich.edu/

[38] *The DETER Project*, http://www.isi.edu/deter/

[39] *The Honeynet Project*, http://www.honeynet.org

[40] Cliff C. Zou *Internet Worm Propagation Simulator*, http://tennis.ecs.umass.edu/ czou/research/wormSimulation.html