PWC: A Proactive Worm Containment Solution for Enterprise Networks

Yoon-Chan Jhi*, Peng Liu†, Lunquan Li‡, Qijun Gu§, Jiwu Jing¶ and George Kesidis‖

*†‡‖The Pennsylvania State University, University Park, PA 16802
Email: {*jhi,‖kesidis}@cse.psu.edu, {†pliu,‡lli}@ist.psu.edu

§Texas State University, San Marcos, TX 78666

¶Chinese Academy of Sciences

***Abstract*—We propose PWC, a proactive worm containment solution for enterprises. PWC can stop - instead of slowing down - an infected host from releasing worm scans as early as after merely 4 scans. Motivated by the observation that a worm uses a sustained outgoing packet rate, PWC gains infection awareness seconds before a signature or filter can be generated. To overcome denial-of-service possibly caused by such smoking signs of infection, PWC develops two new white detection (detecting who are uninfected) techniques: (a) the *vulnerability time window lemma*, and (b) the *relaxation analysis*. PWC is signature-free thus it is immunized from polymorphic worms and timely in containing. PWC is also resilient to containment evading. PWC is not sensitive to worm scan rate, and not protocol specific. Due to white detection, PWC causes minimal denial-of-service. Evaluation based on real traces and worm simulations demonstrates that PWC significantly outperforms Virus Throttle [1] in terms of number of released worm scans, number of hosts infected by local scans, and availability.**

I. Introduction

Computer worms (i.e., malicious self-propagating code) are a significant threat to Internet security. The severity of their damage has been well demonstrated by a set of high-profile Internet wide worm attacks: (a) Within merely 5 minutes, 75,000 SQL Servers were infected by the Slammer worm in 2003: the peak global scan rate was above 55 million scan packets per second; a lot of enterprise networks were deadly congested; and many sites had to be down for recovery. (b) Within 16 hours, about 350,000 hosts were infected by the CodeRed-II worm in 2001. (c) In 2004, the Witty worm infected about 12,000 systems where data stored in certain portions of the disks were destroyed. These high-profile worm attacks have clearly shown that the loss caused by a worm attack against an enterprise (and her business) can be potentially huge (e.g., tens of millions of dollars). Such a severe worm attack can occur at literally *any* point of time under the discretion of the adversary (e.g., hackers, criminals, and terrorists), therefore the risk of worm attacks will not be significantly reduced until highly-effective and highly-practical worm detection and containment technologies are developed.

Since worm infection can spread more rapidly than human response, automated worm detection and containment techniques are essential. In addition, worm containment techniques should be able to handle zero-day (unknown) worms. Enterprise level worm containment has three basic goals: (1) prevent internal hosts from being infected; (2) block outgoing worm scans; (3) minimize the denial-of-service effects caused by worm containment controls.

Many approaches have been proposed to perform the enterprise level worm containment. However, current defenses do not complete four specific requirements which include (R1) *timeliness* in policing worm scans, (R2) *resiliency* to containment evading, (R3) *minimal denial-of-service costs*, and (R4) *being agnostic* to worms' scanning strategy to contain a wide spectrum of worms from uniformly randomly scanning worms to *topologically aware* scanning worms. [2], [3], [4], [5], [6], [7] have limited application of R1, [2], [3], [4], [5], [6], [7], [8], [9], [10], [11], [12] are short of R2, [1] lacks R3, and [3], [4], [5], [7], [11], [12] have limitation in R4.

To overcome above limitations, we propose PWC, a novel proactive worm containment solution for enterprises. The idea of PWC is motivated by two important observations: (O1) If every *infected* host can be immediately disabled from releasing UDP packets or getting outgoing TCP connections connected, the worm will be contained, even if incoming UDP and TCP connections are *still* allowed. (O2) In order for a worm to be *fast* in propagating itself, any infected host must use a sustained faster-than-normal outgoing packet rate.

O1 and O2 indicate that PWC may use a sustained faster-than-normal outgoing connection rate to *be-aware* that a host is infected and the awareness can be gained many seconds before a signature or filter is generated; then the host's outgoing UDP packets and TCP connection attempts can be instantly blocked – instead of being slowed down – to achieve quick, *proactive* containment. By providing novel ways to handle false positives transparently or with minimized impact, PWC allows worm containment systems to react earlier without suffering from high false-positive rates.

To overcome denial-of-service effect that could be caused by false positives (in identifying infected hosts), PWC develops the following two novel *white detection* techniques: (a) PWC exploits a unique *vulnerability window lemma* to avoid false initial containment; (b) PWC uses a *relaxation analysis* to uncontain (or unblock) those mistakenly contained (or blocked) hosts within few seconds, if there are any. Note

This work was supported by NSF and DHS under award number NRT-335241. Jiwu Jing was supported in part by NSFC grant No.60573015.

that existing rate-limiting techniques are worm-slowing-down techniques, while PWC is a *worm-halting* technique. Finally, PWC integrates itself seamlessly with existing signature-based or filter-based worm scan filtering solutions. As soon as a signature or filter is generated, PWC can stop enforcing any new containment controls and unblock all still-being-contained hosts. With a little extension, PWC can utilize the multi-resolution worm detection technique [13] and react against slower worms. In this paper, we will discuss single resolution detection as our focus is on *contain-and-relax* framework.

We have evaluated the cost-effectiveness of PWC using both real world traces and extensive simulation experiments. Our empirical study shows that PWC is significantly outperforming Virus Throttle scheme proposed by Williamson *et al.* [1] in terms of all of the three evaluation metrics: (M1) number of released worm scans, (M2) number of hosts infected by local worm scans, (M3) total denial-of-service time per host.Moreover, the experiments show that PWC is significantly outperforming Hamsa [5] in terms of M1 and M2 with negligible denial-of-service costs. The merits of PWC summarized below show that PWC has taken a major step forward in meeting the requirements aforementioned.

⊙ PWC is *signature free*. Without the need to match a message with a signature or a filter, PWC is *immunized from polymorphic worms and all worm code obfuscation methods*.
⊙ Exploiting an obvious property of scanning worms (i.e., any infected host must use a sustained outgoing packet rate), PWC is *resilient to containment evading*.
⊙ *Timeliness.* PWC may react to worm scans many seconds before a signature or filter is generated.
⊙ PWC is *agnostic* to the scanning strategy of worms since it does not rely on any symptoms caused by specific scanning strategy (i.e., ICMP type 3 messages for failed scans).
⊙ Exploiting the vulnerability time window theorem and the white detection idea, PWC causes *minimal* denial-of-service.
⊙ PWC is NOT protocol specific.
⊙ PWC performs containment consistently over a large range of worm scan rates. PWC is *not sensitive* to worm scan rate.

II. Related Work

Existing worm containment techniques can be roughly broken down into five classes as follows.

Class A: Rate limiting. The idea of Class A techniques is to limit the sending rate of scan-like traffic at an infected host. Virus Throttle proposed by Williamson *et al.* [1] uses a *working set* and a *delay queue* to limit the number of new machines that a host can connect to within unit time. In [14], connection failure rate is exploited, and, in [15], the number of unique IP addresses that a host can scan during each *containment cycle* is leveraged. Class A techniques may introduce longer delays for normal traffic.

Class B: Signature-based worm scan filtering. The idea is to generate the worm signature which can then be used to prevent scans from entering/leaving a LAN/host. Earlybird [2] is an efficient inline solution that integrates flow classification, signature generation and scan filtering. However, it can be easily evaded by polymorphic worms. Polygraph [4] can handle polymorphic worms, but it spends too much time in generating the signature. In [16], [17], [18], signatures are generated out of packets “captured” by a honeypot. However, network-level flow classification techniques used invariably suffer from false positives leading to noise in the worm traffic pool [5]. Although Hamsa [5] is a fast, noise-tolerant solution against network flows, the false negative and false positive of a signature depend on the accuracy of the flow classifier used. In addition, Hamsa and many other Class B solutions are vulnerable to Polymorphic Blending attacks [19]. PWC is signature free - containing worms without using signatures. Accordingly, although PWC cannot prevent worm scans from entering a host, PWC is in general much more resilient to polymorphic worms and worm code obfuscation than Class B techniques, and PWC has much better timeliness.

Class C: Filter-based worm containment. Class C techniques shares the same spirit with Class B techniques except that a filter is a piece of code which is to check a message if it contains a worm. Shield [20] uses host-based filters to block vulnerabilities but these filters are generated manually. Vigilante [7] generates and distributes host-based filters automatically. But, its response time relies on the worm payload size, and some filters can be evaded by code obfuscation based on char shifting or insertion. To achieve high coverage, they need a complicated detection technique such as dynamic dataflow analysis [21], [22]. PWC is not using such filters.

Class D: Payload-classification based worm containment. The idea of Class D techniques is to determine if a packet contains a worm. In [8], [9], [23], a set of anomaly detection techniques are proposed to detect worms. But, they suffer from false negatives or false positives, especially in the presence of code obfuscation. In [10] control flow structures are exploited to detect polymorphic worms, but, off-line analysis is required. In [24], [11], they detect if a data packet contains code or not, but, not all worms propagate through data packets. PWC does *not* do code analysis on payloads.

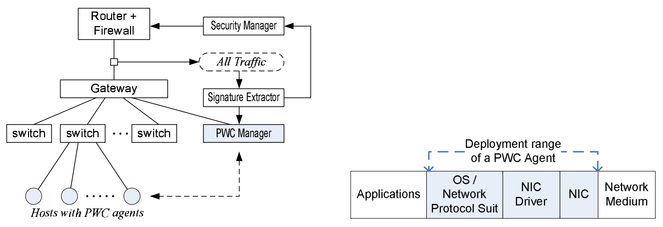
Class E: Threshold Random Walk (TRW) scan detection. In [25], TRW exploits *randomness* in picking destinations to connect to, to detect if a host is a scanner. In [12], hardware implementation is investigated. TRW is suitable for deployment in high-speed, low-cost network hardware, and it is very effective in tackling the common way of worm scanning (i.e., random scanning with high failing likelihood). Attackers may evade TRW, using such attacks as two-sided evasion [12] to which PWC is not vulnerable.

III. PWC Overview

A. Definition and Scope

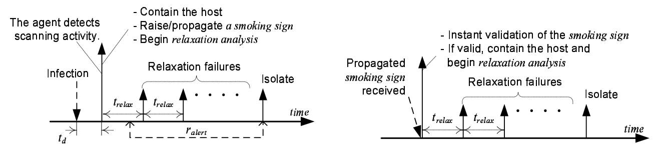
Target Worm. We consider UDP/TCP-based scanning worms, hit-list scanning worms, and topologically aware (or local preferential scanning) worms.

Worm Scan. We classify worm scans in three types: *L-L scans* from an internal(local) infectee to an internal address, *L-R scans* from an internal infectee to an external(remote) address, and *R-L scans* from an external infectee to an internal address.



(a) Deployment diagram (b) Conceptualized host

Fig. 1. The enterprise network protected by PWC.


The agent detects scanning activity.
- Contain the host
- Raise/propagate a smoking sign
- Begin relaxation analysis
Infection
Relaxation failures
Isolate
t_{relax}
t_{relax}
time
t_d
r_{alert}
- Instant validation of the smoking sign
- If valid, contain the host and begin relaxation analysis
Propagated smoking sign received
Relaxation failures
Isolate
t_{relax}
t_{relax}
time

(a) Active containment (b) Passive containment

Fig. 2. Time-line at each host running the PWC agent. Alerts from the conventional worm detectors are raised within the range r_{alert}.

Connection Attempts and Connections. An *outbound connection attempt* is defined as either an outbound TCP SYN or an outbound UDP packet. A *successful outbound connection* is defined as an observation of an inbound TCP SYN-ACK packet. UDP packets are always considered as successful connections. Inbound connections/connection attempts are defined in similar ways. When we mean TCP connections, we use 'TCP connections' explicitly.

Signature Extractors. PWC can be a layer in a multi-layer defense approach. As an example of another layers, we assume an automatic signature generation system(s) is operational in the same enterprise network.

B. Architecture

As shown in Figure 1(a), each host in the protected enterprise network runs a PWC agent which performs detection and suppression of worm scans released from its host. We conceptualize a host running the PWC agent as shown in Figure 1(b). Discussions on implementation issues can be found in Section VII-C. The PWC manager has two roles: first, it distributes authenticated worm alerts reported by a PWC agent to all PWC agents in the enterprise network; second, it is a certificate authority in authentication between each PWC agent and the PWC manager and vice versa. PWC can handle multiple simultaneous smoking signs raised by different worms in one contain/relax procedure.

Before getting into details, we briefly summarize operations of PWC system, from A to G, in an event-driven manner following the time line in Figure 2. We will discuss the details on the following operations in section IV one by one.

(1) When a PWC agent detects a scan activity. The PWC agent takes the following actions in order: (a) raises a smoking sign; (b) initiates containment on its host, which is called *active containment*; (c) reports the smoking sign to the PWC manager; (d) starts *relaxation analysis* operations on its host. c is to let other PWC agents be aware of the situation and check their hosts if they are infected. d is required since the agent needs to detect sustained rate of connection attempts to distinct destination addresses, to determine the host is infected.

(2) When PWC manager receives a smoking sign. The PWC manager propagates the smoking sign to all other agents. The frequency of smoking sign propagation is controlled by PWC. Note that the focus of this paper is not on the underlying mechanism of the smoking sign propagation.

(3) When a PWC agent receives a smoking sign. The PWC agent (a) performs *vulnerability window analysis* (see Section IV) to see whether its host is possibly infected or not; (b) ignores the smoking sign if no evidence of possible infection is found; (c) otherwise, initiates containment on its host, which is called *passive containment*; (d) and immediately starts *relaxation analysis*. a and b minimize availability-loss possibly cased by excessive passive containments.

(4) When a PWC agent is performing relaxation analysis. During the relaxation analsysis, the PWC agent calculates the rate of outbound connection attempts to distinct IP addresses, and checks if the host shows sustained connection rate or not. The relaxation analysis is limited to t_{relax} seconds.

(5) When a PWC agent completes relaxation analysis. Based on the result of (4), the agent relaxes or continues the containment. If the agent relaxes the containment, it will repeat above operations from (1). If the agent continues the containment, it will repeat (4) once more. After F relaxation failures, the agent will isolate its host and report to the PWC manager for further handling. We observed no isolated uninfected host through number of experiments with $F = 30$ and $t_{relax} = 1$.

(6) When signature extractors identify new signatures. The signatures are reported to the PWC manager. The PWC manager relays it to a security manager so that it may be installed into firewalls to block inbound (or outbound) malicious messages. At the same time, the signature is propagated to all PWC agents and will be installed in the agents' embedded packet filters. The packet filters are to reduce smoking signs raised by identified malicious messages.

IV. The PWC Approach

PWC consists of three major phases: *smoking sign detection* (section IV-A), *initial containment* (section IV-B, IV-C), and *relaxation* (section IV-D) phases.

A. Raising Smoking Signs

1) Smoking Signs and Active Containment: Smoking signs are require to be raised early, but not necessarily to have an extremely low false-positive rate. This important characteristic allows PWC agents to contain possibly infected hosts swiftly without hesitation while requiring consequent relaxation phases to resolve the false positives. Since, to survive in the wild, the worm must replicate itself to at least another victim before being contained, the worm naturally sends infectious messages to as many distinct destination addresses as it can. Therefore, abnormal growth in the number of distinct addresses at infected hosts has been in the literatures [2], [26], [27]. We observed in a 24-hour Auckland-IV trace [28] that majority of the hosts connected to less than 15 distinct IPs/sec, and only few of them connected to 20-25 distinct IPs/sec. In our lab traces, the rates of distinct destination addresses were no more than 5 IPs/sec. In contrast, the CodeRed-I, for example, probes more than a hundred unique IPs per second.

Algorithm 1 Smoking-sign detector

```
1  ▷ $inconhist$, $outconhist$: lists of recent in/outbound
                           connection attempts
3  ▷ $dsthist$: set of known destination IP addresses
4  ▷ $srchist$: set of known source IP addresses
5  ▷ $pkt$: a TCP SYN or UDP packet to be sent
6  procedure ON_OUT_CONNECTION($pkt$)
7  begin
8      if (host is contained) then
9          ON_OUT_CONNECTION_CONTAIN($pkt$) and return;
10     if ($pkt.dst\_ip \in dsthist \cup srchist$) then
11         Process $pkt$, and return;
12     Insert $pkt.time$ to $outconhist$;
13     $r$ := rate of the most recent $\Delta$ elements in $outconhist$;
14     if ($r > \lambda$) then
15     begin
16         Start active containment;
17         Report a smoking-sign to PWC manager;
18     end;
19 end.
```

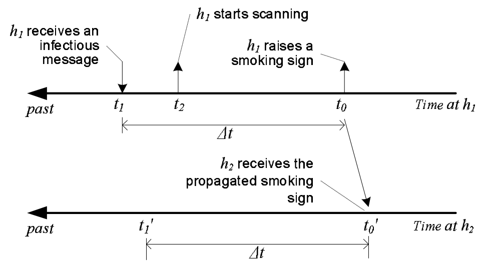
Algorithm 1 shows how PWC agents raise smoking signs and initiate active containment (Figure 2a). ON_OUT_CONNECTION_CONTAIN() in line 9 follows the description in Section IV-C and Section IV-D. On every connection attempt to a new IP address, a PWC agent calculates the rate r based on the most recent Δ elements in *outconhist, the outbound contact history* which is a list of the time-stamps of recent outbound connection attempts made to new addresses. If r exceeds the threshold λ, the PWC agent raises a smoking sign, initiates active containment on its host, and reports the smoking sign to the PWC manager.

2) Smoking Sign Propagation: The following message carries the smoking sign reported to the PWC manager: [t_{sent} + t_d + the agent's IP]. t_d, the detection latency to be used in Section IV-B1, is defined as $t_{sent} - t_{in}$. t_{sent} is the current time and t_{in} is the time-stamp of the latest successful inbound connection made before the Δ time-stamps referenced in calculating r. To prevent possible bandwidth saturation caused by worms from interfering with the smoking sign report, the agent reports the smoking sign after containing its host.

Any smoking sign detected at a host imply the possibility of hidden infectees in the network. To proactively block the hosts that are infected but not detected, the PWC manager shares reported smoking signs with all the agents in the network through *the smoking sign propagation*. Information propagation techniques for cooperative defenses against Internet worms are in the literature [29], [30]. In this paper, we assume a technique for the PWC manager to propagate smoking signs.

The receivers of either reported or propagated smoking signs would discard the smoking signs if t_{sent} is too old. To prevent *forged smoking sign injection*, all the messages between PWC agents and the manager should be authenticated using RSA. Details are discussed in the Security Analysis Section (Section VII-A). The behavior of a PWC agent after receiving a propagated smoking sign is described in Section IV-B.

To avoid denial-of-service and overwhelming traffic, smoking signs will not be reported to the PWC manager if the time elapsed since the most recently received smoking sign is less than the relaxation analysis duration t_{relax}. In this case, we consider all the suspicious hosts in the network are already contained by the previously received smoking sign and are in the relaxation analysis. The PWC manager also applies similar restriction. Therefore, the smoking sign propagation rate is limited to $\frac{1}{t_{relax}}$ times/sec.



h_1 receives an infectious message
h_1 starts scanning
h_1 raises a smoking sign
past
t_1
t_2
t_0
Time at h_1
Δt
h_2 receives the propagated smoking sign
past
t_1'
t_0'
Time at h_2
Δt

Fig. 3. Vulnerability Window

3) Reducing False Smoking Signs: In our network, false positive smoking signs were mainly caused by the applications that send excessive small UDP packets to many distinct destinations (e.g., P2P file sharing and mDNS protocols). To reduce such false smoking signs, we ignore outbound UDP packets that are shorter than 200 bytes. A scanning worm inherently has a byte-sequence to exploit vulnerability, code to select victims, code to send crafted infectious messages, and at least one loop. Even if a worm uses smaller packets to probe vulnerabilities, a worm must have a lower limit in size. For example, among the 100 UDP-based worms obtained from Symantec's Viruses & Risks Search, the smallest payload length was 376 bytes (SQL Slammer).

B. False-Containment Avoidance

A propagated smoking sign makes every agent in the network start passive containment (Figure 2b). On receiving a propagated smoking sign, the agent validates the smoking sign first, which we named *false-containment avoidance*. Note that passive containment is a *proactive* action taken on a host that is not suspicious to local PWC agent's knowledge. Therefore, any propagated smoking sign can be ignored if the receiving agent ensures that its host is not infected. A way to do this is the *vulnerability window analysis* which yields instant decision at each PWC agent on receiving a propagated smoking sign. The decision results in either of *SAFE* and *UNSAFE*, where *SAFE* means the PWC agent can safely ignore the smoking sign, and UNSAFE means the agent should not.

1) The Vulnerability Window Analysis: Consider PWC is fully deployed in an enterprise network. Let us assume all the PWC agents configured with the same parameters since, typically with many organizations, most hosts within the same enterprise network would have similar ability to send packets. Let us assume that infected host h_1 raises and propagates a smoking sign through the PWC manager. Given that h_2 is one of recipients of the propagated smoking sign, let us depict the timeline of the propagation in Figure 3 where,

i. t_1 at h_1 is the time of the last successful inbound connection before releasing the first scan.
ii. t_2 at h_1 is the time when (potentially) the first scan is released.

iii. t_0 at h_1 is the time when a smoking sign is raised.
iv. Δt is equal to $(t_0 - t_1)$
v. t'_0 at h_2 is the time of receiving smoking sign from h_1
vi. t'_1 at h_2 is equal to $(t'_0 - \Delta t)$
vii. t_{in} at h_2 is the time of the last successful inbound connection.

Let us assume (a) h_2 is susceptible to the same worm as h_1 has; (b) h_2 is not contained at t'_0; (c) $\Delta t < t_{relax}$; (d) h_1 and h_2 have similar CPU/NIC performance. (a) and (b) are considered to be true, PWC should be configured to hold (c), (d) is generally true in an enterprise network. We do Vulnerability Window Analysis by testing the following hypothesis: (e) the connection attempt made at t_{in} was infectious. The merit of this analysis is that if the hypothesis is proven False, $h2$ can safely ignore the smoking sign and avoid containing an innocent host. To see if the hypothesis is False, we assume the hypothesis were True, then we prove by contradiction.

To determine whether h_2 needs to be contained or not at time t'_0, we must consider the following cases (1) and (2).

(1) $t_{in} < t'_1$: If hypothesis (e) were True, h_2 should have been infected at t_{in}, and PWC agent at h_2 must have raised a smoking sign within the time window $[t'_1, t'_0]$ and become contained. From (b), h_2 is not contained at t'_0, thus we can conclude h_2 was not infected at t_{in}. Because h_2 has never been connected since t_{in}, h_2 is considered to be SAFE.

(2) $t_{in} > t'_1$: h_2 should be considered to be UNSAFE, for we cannot reject hypothesis (e).

Therefore, we have Lemma 1, *vulnerability window lemma*.

Lemma 1: At t'_0, if h_2 receives a propagated smoking sign (t_0, t_d, h_1), h_2 can ignore the smoking sign and skip passive containment if the following assumptions hold:

i. $t_{in} < t'_0 - t_d$
ii. h_2 is susceptible to the same worm as h_1 has.
iii. h_2 is not contained.

Lemma 1 can be extended to handle multiple kinds of worms by taking the larger t_d when smoking signs report different t_d's. Although a worm can evade passive containment by having a delay before starting scanning, the worm cannot successfully spread out since local PWC agent will initiate active containment after monitoring the first Δ scans.

A limitation of vulnerability window analysis is that any inbound connection attempt within the vulnerability window makes the vulnerability window analysis result in *UNSAFE*. The result is affected by two factors: first, frequent legitimate inbound connections; second, large vulnerability window Δt. We will introduce two heuristics to address these limitations and will see how often the vulnerability window analysis would raise false positives with selected Δt. From the definition of t_d in Section IV-A2, the largest Δt can be approximated as $\frac{\Delta}{\lambda}$ seconds when α is zero. $(7, 4)$ and $(7, 10)$, the two pairs of (λ, Δ) that we configured based on real trace experiments, yield $\Delta t =$ 0.57 and 1.43 seconds respectively.

2) Traffic Filter for Vulnerability Window Analysis: To make the vulnerability window analysis resilient to noise (legitimate traffic), we set up two heuristics to sift out meaningful traffic within the vulnerability window. The heuristics are:

⊙ **H1**: Even a fast worm scanning 8,000 IPs/sec with 50% local preference would take more than 16 seconds to scan entire /16 local network. Thus, we regard redundant connection attempts from the same IP address incoming within H_t seconds as noise, and reduce them leaving only the first one.
⊙ **H2**: Eliminate inbound UDP packets whose payloads are shorter than H_l bytes. PWC uses $H_l = 200$ as we discussed in Section IV-A3

We could reduce 96% of the legitimate inbound connection attempts appeared in our lab PC traces by H1 and H2 with $H_t = 10$ and $H_l = 200$. In addition, on the same traces, we calculated P[N=0], the probability that vulnerability window at a certain time point may not include legitimate inbound connection attempts. Although we do not show the result due to the limited space, P[N=0] when $\Delta t = 0.57$ seconds was above 95% and when $\Delta t = 1.43$ seconds was above 90%.

C. How We Contain a Host

During active or passive containment, the agent prohibits its host from connecting to other hosts. Inbound connections and already established sessions are allowed to proceed.

Containment should handle two types of packets which indicate outbound connection attempts: outbound UDP and outbound TCP SYN packets. During the containment, a PWC agent first tries buffering the connection attempts, to forward them when the containment is relaxed. The buffered connection attempts will be dropped with appropriate handling if the buffer becomes full or if the packets are delayed for longer than predefined timeout (up to a couple of seconds). Meanwhile, PWC needs a special handling to integrate itself seamlessly with other network-based signature identification and filtering techniques [2], [3], [4], [5]. When a PWC agent buffers a connection request, it forwards a copy of the packet if the destination address is not in the same enterprise network. Since the copy of connection request should not reach the destination host, the PWC agent replaces the TTL value with the number of hops to the border of network. Given the address of the border router, the agent can measure exact number of hops, using the same method as TRACEROUTE does [31]. Thus, the signature extractor can see worm scans as if the sources were not contained, while the scan from the contained host cannot reach the victims. To prevent congestions on internal paths, the rate of the copy-forwarding must be controlled.

D. Containment Relaxation Analysis

During the containment, a PWC agent maintains dst, the number of distinct addresses to which its host has initiated connection attempts, to see if the host shows sustained rate exceeding λ. We call this analysis *relaxation analysis* since the goal is to relax mistakenly contained hosts. Relaxation analysis for a containment initiated at time $t_{contain}$ monitors the host for at least t_{relax} seconds. The connection rate r_{relax} updated at the end of the relaxation analysis is defined as $\frac{dst}{t_{last\_conn} - t_{contain}}$, where t_{last_conn} is the timestamp of the first outbound connection attempt initiated after $t_{contain} + t_{relax}$. If r_{relax} is lower than λ, the containment

TABLE I
NOTATIONS USED IN EVALUATION.

Symbol	Description
$\|x\|$	The size of x.
V	The number of vulnerable hosts in the enterprise network.
λ	Smoking sign threshold (unique destinations/sec)
Δ	The sample size to calculate r.
nMI	The number of mistakenly isolated uninfected hosts.
rAC	The rate of active containments at a host.
rPC	The rate of passive containments at a host.
nI	The number of infected hosts in the network.
fI	$\frac{nI}{V}$
nI_0	The number of initially infected hosts.
nES	The number of escaped scans.
rS	Worm scan rate.
L	Worm's local preference (0.0 for uniformly scanning worms).
rD	Average delay per connection request at a host.
WIL-x-y	Williamson's Virus Throttle with $\|\text{working set}\| = x$ and $\|\text{delay queue}\| = y$.
PWC-x-y	PWC with $\lambda = x$ and $\Delta = y$.

should be relaxed. Otherwise, relaxation analysis should be performed again. By spanning the calculation of r_{relax} over consecutive relaxation analyses, we can avoid evasion attempts by such worms that periodically scan at a burst rate [32]. F successive failures in relaxing containment will let the host isolated from the network.

V. Experiment Setup

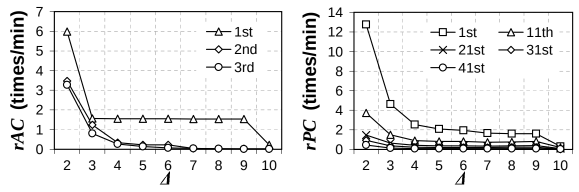
Symbols and notations used in the following sections are described in table I. Through extensive simulations on enterprise-level real traces, we have evaluated (1) cost-effectiveness of PWC; (2) effect of collaboration; and (3) impact of partial deployment. We also implemented a prototype PWC agent[1] to study the impact on local P2P traffic. We have used the following three metrics through out the evaluation:

- (M1) number of hosts infected by local worm scans.
- (M2) number of released worm scan packets.
- (M3) total denial-of-service time per host.

We evaluate PWC against two existing techniques, Williamson's Virus Throttle [1] and Hamsa [5] in terms of each metric. Virus Throttle generates false positives on seven hosts in the background traffic, thus we set up another configuration WIL-5-1500 besides the default WIL-5-100. WIL-5-1500 is the most conservative configuration that does not raise false positives with the tested normal traffic. We deployed Hamsa at the border of the enterprise network in the simulator. Hamsa starts generating signatures when the suspicious pool size reaches 500 and the signature extraction takes 6 seconds [5].

⊙ **Hypothetical Enterprise Network**: The enterprise network simulations assume /16 local address space and 13,000 hosts with $V = 6,500$. We assume no inbound scans from external infectee, for PWC is an unidirectional worm containment approach. Also, Round-Trip-Time (which is typically less than 1 ms) within the same enterprise network is ignored.

⊙ **Background Traffic**: To configure parameters and to render normal traffic, we have used a 24-hour trace of the Auckland-IV traces [28] collected in 2001. The traces collected at the border of the University of Auckland do not contain local-to-local traffic. The omitted traffic would not affect the



(a) rAC at each host. The chart shows values at top three ranked hosts for each Δ.

(b) rPC at five hosts among the top ranked hosts. The 41st host is the baseline of top 2%.

Fig. 4. The active/passive containment rate (rAC, rPC) for different Δ.

experiment results since the observation on our own local network running PWC agent prototype showed that (1) the local-to-local inbound and outbound connections implied high locality which could be filtered by $inconhist$ and $outconhist$; and (2) the burst rate of normal outbound connection attempts did not sustain. In addition, the omitted traffic will also affect existing techniques being compared with our system.

⊙ **Test Worms**: Three types of test worms include (T1) randomly uniformly scanning worms, (T2) 0.3 local preferential scanning worms, and (T3) 0.5 local preferential scanning worms. T2 and T3 worms give idea of PWC's effect on the local preferential scanning worms in real world. For example, the CodeRed-II worm scans the same /8 network with 50% probability and scans the same /16 network with 37.5% probability. The Blaster worm picks the target within local /16 network at a probability of 40% and a random IP at 60%.

VI. Evaluation

A. Tuning Parameters

λ and Δ need to be tuned based on normal traffic, and both parameters are critical to the effectiveness of PWC. We used an Auckland-IV trace to render the normal traffic.

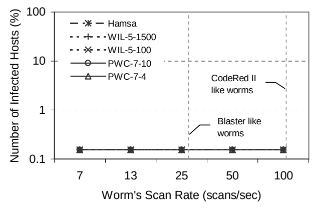
1) The Smoking Sign Threshold: The criterion that we used for a good λ was the number of mistakenly isolated hosts nMI which was, in other words, the false positives that relaxation analysis could not handle. We calculated nMI varying λ and Δ, on a 24-hour long normal traffic. Given that $\Delta = 5$ to reduce the effect of false alarms caused by Δ, we observed $nMI > 3$ when $\lambda < 7$. When $\lambda \geq 7$, nMI was zero even with $\Delta = 2$ for the most aggressive configuration.

2) The Size of the Outbound Contact History: Small Δ enables rapid containment while sacrificing accuracy. Consequent inessential containment would reduce availability.

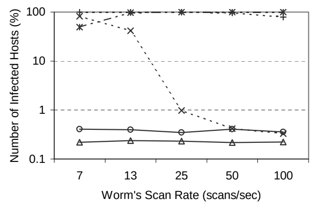
We ran PWC over the normal traffic for 24 hours. For each host, we calculated rAC and rPC, per-minute rates of active and passive containments caused by false smoking signs, varying Δ. Since the vulnerability window analysis would discard some propagated false smoking signs, rPC at each host would be the rate at which the vulnerability window analysis could not reject propagated false smoking signs.

In the result shown in Figure 4, rAC and rPC were stable in the range where $4 \leq \Delta \leq 9$. rPC was less than once in ten minutes at more than 98% of entire hosts when $\Delta = 4$, and more than 99% when $\Delta = 10$. Based on the results, we could set Δ to 4 for conservative and 10 for the less conservative yet more accurate configurations.

[1]Current prototype does not implement collaboration.

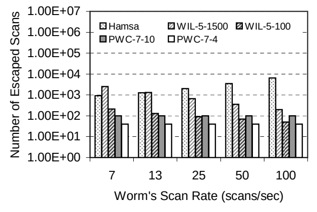


(a) T1 worms ($L = 0.0$)

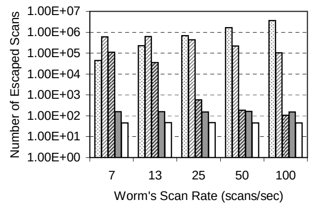


(b) T3 worm ($L = 0.5$)

Fig. 5. The number of infected hosts for local-preferential scanning worms with various scan rates. $V = 6,500, nI_0 = 10$



(a) T1 worms ($L = 0.0$)



(b) T3 worms ($L = 0.5$)

Fig. 6. The number of escaped scans for different worm's scan rate. $V = 6,500, nI_0 = 10$

3) The Relaxation Analysis Duration: The relaxation analysis duration t_{relax} affects the overall time in which each host would be contained. We configured t_{relax} to minimize contained duration, based on simulations (PWC-7-4 with no worm) through the busiest four hours of the normal traffic.

Let us denote by ϕ_i the sum of $\phi_{i,j}$ at host i, where $\phi_{i,j}$ represents the length of the j^{th} containment at host i. We calculated the distribution of ϕ_i for various t_{relax}. The result suggested smaller t_{relax} reduced ϕ_i. False negatives in relaxation analysis (or false UNSAFE) will be resolved in the next relaxation analysis, therefore t_{relax} should be a small value that minimizes containment-relaxation false positives. We set t_{relax} to be 1 second, where ϕ_i at 99.9% of hosts were below 32 seconds which was 0.2% of the simulated time.

B. Performance Evaluation

We evaluated worm containment performance of PWC assuming worm outbreak in the enterprise network with fully deployed PWC. We deliberately set up 10 initially compromised hosts to stimulate local infection.

1) M1 (Local-to-Local Infection Rate): The most significant contribution of PWC is the suppression of local-to-local worm infection. As in Figure 5, PWC successfully suppressed local-to-local infection by local-preferential scanning worms, even under an extreme condition where $nI_0 = 10$, $V = 50\%$.

2) M2 (Escaped Worm Scans): A successful worm containment strategy must minimize the number of scans that escapes the perimeter of defense during the delay when the containment system detects the enemy and prepares its weapon (i.e., signatures). We measured the number of escaped scans for each of PWC, Virus Throttle, and Hamsa, until the worm propagation was completely stopped. Figure 6 shows PWC outperformed Virus Throttle and Hamsa in terms of M2, the number of escaped scans. While Virus Throttle and Hamsa performed better for the faster worms and the slower worms respectively, PWC showed consistent performance for all the

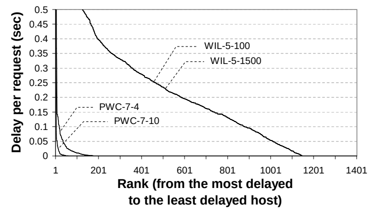


Fig. 7. Sorted *average delay per connection request* observed at each host.

tested scan rates. As the worm scanned local address space more aggressively, the performance gap between PWC and other techniques became more significant. Although WIL-5-100 performed better than PWC-7-10 for the worms faster than 25 to 50 scans/sec, WIL-5-100 isolated 7 hosts due to the false positives. We observed no naive host had been isolated by PWC during the simulations on M2.

3) M3 (Total Denial-of-Service Time Per Host): We compared PWC in terms of availability-loss that the containments caused by false smoking signs introduced. Please note Hamsa does not introduce the availability-loss, thus, we compared PWC with Virus Throttle only.

To compare availability-loss, we calculated rD, the average delay per request at each naive host, running on the same 24-hour-long background traffic. As shown in Figure 7, PWC significantly outperformed WIL-5-100 and WIL-5-1500 in terms of M3. Due to the long delay queue, WIL-5-100 and WIL-5-1500 delayed outbound connection requests for couples or even tens of seconds *in average* at several hosts while the maximum rD was 0.95 sec/request for PWC-7-4 and 0.5 for PWC-7-10. Variations were 0.0016 and 0.0002 for PWC-7-4 and PWC-7-10 respectively. Per-request delays for 99.80% of the hosts in the PWC-7-4 experiment were below 0.37 sec, and for 99.99% in the PWC-7-10 experiment were below 0.29 sec.

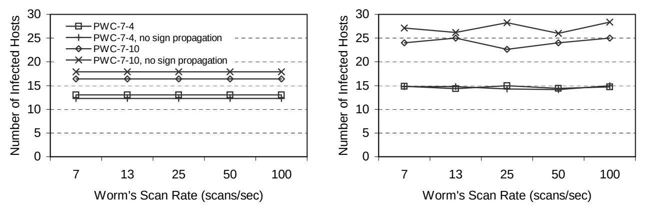
C. Impact of Smoking Sign Propagation Delay

To see the impact of smoking sign propagation delay, we studied the worst case. In particular, for each of PWC-7-4 and PWC-7-10, we set up two different PWC systems: one propagated smoking signs in LAN speed, while the other propagated *no* smoking signs. In each system, we infected 10 hosts with each of worms T1, T2, and T3. We performed 100 simulations in each experiment and compared the performance in terms of M1 (Figure 8) and M2 (Figure 9).

For T1, we observed no significant impact of smoking sign propagation delay because all the infected hosts were contained before infecting any local victims. Similar phenomena were observed in the cases of PWC-7-4 with T2 and T3. However, in the cases of PWC-7-10 with T2 and T3, smoking sign propagation improved performance by 4.6-19.8% and 4.3-10.6% in terms of M1 and M2 respectively. The results suggested the performance improved as worms' preference to the local addresses increased.

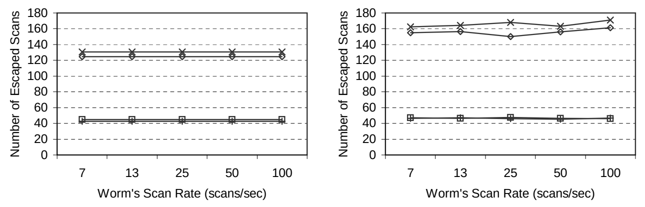
D. Impact of Partial Deployment of PWC

We performed experiments on different deployment scenarios where 40%, 60%, 80% and 100% of hosts in the enterprise network were running PWC agents. Compared with fully-deployed PWC, the performance of partially-deployed PWC



(a) T2 worms ($L = 0.3$) (b) T3 worms ($L = 0.5$)

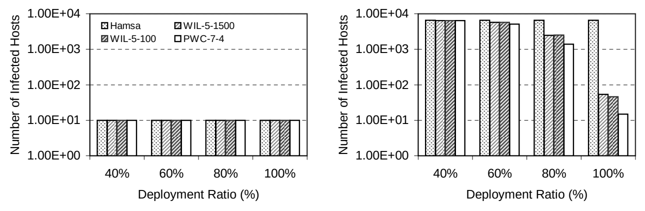
Fig. 8. Smoking sign propagation effect in terms of the number of infected hosts. V = 6,500, nI_0 = 10



(a) T2 worms ($L = 0.3$) (b) T3 worms ($L = 0.5$)

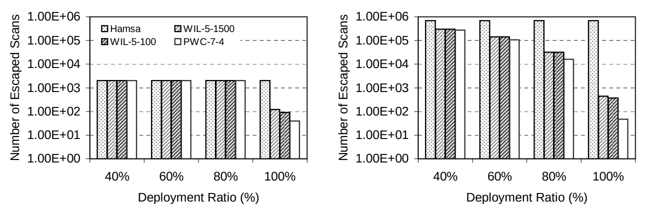
Fig. 9. Smoking sign propagation effect in terms of the number of escaped scans. V = 6,500, nI_0 = 10

would degrade in linear speed with respect to the deployment percentage. We also compared partially-deployed PWC with fully-deployed Hamsa and partially-deployed Virus Throttle. Our comparison focused on the time window Π which starts when the worm attack is mounted and ends when Hamsa generates the signature, for PWC will (typically) terminate itself as soon as the signature for the worm is generated. We first infected 10 unprotected hosts with a worm sending 25 scans/sec. Then we defined Π by running Hamsa. Finally, we measured M1(Figure 10) and M2(Figure 11) of Virus Throttle and PWC during Π. Overall, partially-deployed PWC-7-4 performed substantially better than fully-deployed Hamsa; and PWC-7-4 performed better than or equal to Virus Throttle in all the partial deployment scenarios. Virus Throttle's performance could be worse since it would delay Hamsa's suspicious traffic collection phase while PWC would not.



(a) T1 worms ($L = 0.0$) (b) T3 worms ($L = 0.5$)

Fig. 10. The number of infected hosts for different deployment ratio (in log-scale). rS = 25 scans/sec, V = 6,500, nI_0 = 10



(a) T1 worms ($L = 0.0$) (b) T3 worms ($L = 0.5$)

Fig. 11. The number of escaped scans for different deployment ratio (in log-scale). rS = 25 scans/sec, V = 6,500, nI_0 = 10

E. *Impact on P2P Traffic*

We tested a prototype PWC agent with eMule 0.47c, a P2P application sharing files over the ed2k and Kad (server-less) networks. The prototype was installed in a Winsock LSP layer of a PC running Windows XP Professional SP2 connected to Internet through a cable modem. During 9.35 hours of experiment, eMule shared 494 files (130.31MB in average), with unlimited uploading and downloading speed (effectively 60-70 KBytes/sec and 120-130 Kbytes/sec respectively). We observed logarithmic increase of 3,806 distinct peers were monitored in 10,990 connection attempts. eMule released excessive UDP packets shorter than 150 bytes in the Kad network mainly to search sources and peers.

To eliminate eMule traffic in raising smoking signs, our prototype leveraged the following observations on eMule traffic: (a) short UDP packet size and (b) large but limited number of distinct destinations. As the result, eMule had been contained 11 times during the first 1.08 hours and never been contained afterwards. Only 0.28% of the outbound connection attempts were delayed (but not discarded) 1.35 seconds in average (maximum 2.09 seconds).

VII. Security Analysis

A. *Smoking Sign Injection Attacks*

There are two types of smoking sign injection attacks: a spoofed smoking sign injected from an external host; a forged smoking sign from a internal host compromised by an attacker.

1) Smoking Sign Injection from an External Host: In this attack mode, we assume the attacker has no knowledge about the authentication keys used in smoking sign propagation. A proper firewall configuration that ensures no incoming packet has any internal source address can filter simple injection of bogus smoking signs spoofing internal source address. However, a sophisticated external attacker may use IP tunnel. Thereby, PWC uses a public-key authentication scheme to authenticate each PWC agent to the PWC manager and vice versa. The scheme is scalable in that (1) PWC agents only need to verify the validity of the PWC manager's certificate and (2) the PWC manager knows whether or not an agents certificate has been revoked as the PWC manager issues certificates to PWC agents. Although public-key authentication operations may consume CPU cycles, fast RSA [33] in current hardware can authenticate within 100 μsec and enterprise hosts typically have enough CPU power. As shown in Figure 9 and Figure 10, the impact of small delay on PWC is negligible. (Note that symmetric-key authentication is not as much manageable regarding the number of PWC agents.)

2) Smoking Sign Injection from an Internal Host: In this attack mode, we not only upgrade the attacker's capability so that he may "steal" the key from a compromised host, but also assume *insider threat*. Under a public-key authentication scheme, the attacker can forge a smoking sign either by invoking the signing subroutine of the (local) PWC agent, or by "'stealing" the private key used by the compromised host.

The First Attempt As we will review shortly in Section VII-C, how a PWC agent is implemented determines

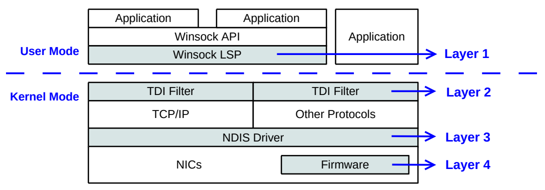


Fig. 12. Windows Network Architecture Diagram

the ability of an internal attacker to "fool" the PWC agent. Because a PWC agent accepts and processes only restricted part of packet headers, the agent program itself is less likely to have buffer related vulnerabilities. When implemented in software, modern computer architecture and operating systems practices also provide a variety of memory protection measures that should be applied to protect PWC agents. By taking these protective measures, the first attempt would have a high probability in crashing the system instead of "fooling" the system. More conservatively, we could implement a PWC agent as a piece of firmware inside a NIC card.

The Second Attempt requires a lot of knowledge on where and how the private key is stored. This threat leads us to the fundamental problem of runtime key protection which is well studied in the literature. If the private key has to be stored in the memory, we have two defenses: use of key obfuscation techniques to store keys encoded, or use of key partitioning to break the key into several parts and store them in different memory locations. Another way is to use hardware modules (such as Trusted Platform Module) that store the private key and never disclose it to outside [34]. The key will be safe if the attackers do not know how to use the module.

B. False Positive Smoking Signs

Three possible reasons may cause a PWC agent to raise false positive smoking signs: a compromised agent/host which is *already* addressed in the previous section; a burst rate in connection attempts which is *already* handled by smoking sign detector and relaxation analysis; special applications such as proxy servers – note that P2P/VoIP/Audio/Video streaming clients and Instant Messengers raise few or no false smoking signs by Algorithm 1.

Proxy servers that connect external clients with internal servers scarcely raise false smoking signs once internal servers start occupying $dsthist$. However, proxy servers serving internal clients often connect to various external servers and may cause false smoking signs. In this case, instead of disabling PWC agents at the proxy servers, we can apply a proper firewall configuration to deny any requests from external clients, thus preventing external worms. As shown in Figure 5, internal worms hardly reach internal hosts including proxy servers before being contained.

Finally, for applications/services that seldom generate bursts in connection requests, those *seldom* experienced extra delay will be minimal as in Figure 7.

C. Worms May Bypass/Disable the Agent after Compromising the Host

Let us assume that we implement a PWC agent in one of the Layers in Figure 12. Sophisticate worms may try the following attempts to neutralize local PWC agent after they successfully break into a host: attempts to *bypass*; or *disable*.

Bypass Attempts Security measures implemented in Layers 1 and 2 are vulnerable to the bypass attempts of a worm that can directly access Layer 3 or Layer 4 interfaces. However, it is not feasible for a worm to bypass Layer 3 since, in order to spread among asymmetric systems, the worm must be able to access variety of NICs directly without using drivers. Bypassing Layer 4 is also impossible since the worm must use NIC. Thus, if the PWC agent is implemented in either one of Layers 3 and 4, it should be very difficult or even impossible for the worms to bypass the agent.

Disable Attempts When a worm exploits one of kernel-mode vulnerabilities, it could have power to (a) unload any programs including drivers; and (b) access full address space. Power (a) allows the attackers to write a worm that propagates freely after unloading PWC agents. However, if the drivers that are necessary for propagation are unloaded, the worm will be contained. Thus, PWC agents can be embedded in those *necessary drivers* such as Layer 3. Note that there are as many NIC drivers as the number of NIC products. Assuming that the vendors embed PWC agents in their drivers, it could be almost impossible for a worm to try power (b) to disable the agent without crashing the driver: although the worm might disable a few drivers, it becomes a partial deployment scenario at most.

D. Other Counterattacks

1) *Poisoning Attacks:* Attackers may try *UDP-flooding attack* in which an internal or external attacker sends excessive UDP packets to a protected host to keep the recipient's vulnerability window analysis yielding UNSAFE. Then, any propagated smoking sign received at the host will lead to passive containment. However, this attack cannot be successful since PWC controls the frequency of propagated smoking signs. Moreover, attackers cannot mimic propagated smoking signs as in Smoking Sign Injection Attacks.

2) *Replacement Attacks:* Replacement attacks are to overwrite (or erase) PWC agents stored in file systems. Worms must use system-calls or BIOS service routines to access file systems. So, we may have the system-calls deny attempts to overwrite PWC agents. BIOS also can help by storing the PWC agent in a restricted area on a disk and denying any unauthorized write-access to the area. In addition, *Microsoft Windows Vista* and *Linux* support Trusted Platform Module to protect file systems from unauthorized changes.

3) *Wait-before-Scan Attack:* A worm may try waiting a prolonged period before starting scanning to evade passive containment. This attempt cannot let the worm successfully spread out because the PWC agent at the host will initiate active containment after the worm releases the first Δ scans.

VIII. Discussions

A. Applicability

Besides uniformly scanning worms, PWC can successfully suppress topologically aware worms, hit-list worms, flash worms, polymorphic worms, metamorphic worms, etc. that scan more than λ new addresses per second. PWC agents are light-weight so that they can be implemented in either way of hardware or software component. Other worm defense measures can be run in parallel with PWC since PWC agents can still forward large part of malicious messages during containment. PWC guarantees those forwarded malicious messages will not infect any host. Finally, PWC allows P2P traffic.

B. Limitations

PWC also has several limitations. First, as a host-based approach, PWC requires majority of internal hosts to run PWC agents. However, the performance degradation in various partial deployment scenarios is not worse than existing techniques. Second, proxy servers of specific type need to be protected in an alternative way as mentioned in Section VII-B. Third, as a certificate authority, the PWC manager must be running in a highly secured host. However, without the PWC manager, the performance of PWC is still acceptible as we discussed in Section VI-C. Finally, during the containment, a PWC agent may experience stalled-scan problem in which a worm's scanning rate is slowed down. This could let the PWC agent relax an infected host after performing a couple of rounds of relaxation analysis. However, this is a problem limited only to the TCP-based worms scanning in a *synchronous* manner, and PWC can still slow down those worms.

IX. Conclusion

In this paper, we proposed PWC, a proactive worm containment solution for enterprises. With aggressive containment and subsequent relaxation analysis based on two novel white detection techniques, PWC could *stop* an infected host as early as after merely 4 to 10 scans were released while minimizing denial-of-service effects. Evaluation based on real traces and extensive worm simulations demonstrated that PWC significantly outperformed Virus Throttle [1] in terms of all of three metrics and Hamsa [5] in terms of local-to-local infections and local-to-remote infections. In partial deployment experiments, PWC outperformed Virus Throttle.

References

[1] J. Twycross and M. M. Williamson, "Implementing and testing a virus throttle," in *USENIX Security*, August 2003.

[2] S. Singh, C. Estan, G. Varghese, and S. Savage, "Automated worm fingerprinting." in *OSDI*, 2004, pp. 45–60.

[3] H.-A. Kim and B. Karp, "Autograph: Toward automated, distributed worm signature detection," in *Proceedings of the 13th Usenix Security Symposium*, August 2004.

[4] J. Newsome, B. Karp, and D. Song, "Polygraph: Automatic signature generation for polymorphic worms," in *IEEE Security and Privacy Symposium*, May 2005.

[5] Z. Li, M. Sanghi, Y. Chen, M. Y. Kao, and B. Chavez, "Hamsa: Fast signature generation for zero-day polymorphic worms with provable attack resilience," in *Proceedings of IEEE Symposium on Security and Privacy*, 2006.

[6] Z. Liang and R. Sekar, "Fast and automated generation of attack signatures: A basis for building self-protecting servers," in *Proc. 12th ACM Conference on Computer and Communications Security*, 2005.

[7] M. Costa, J. Crowcroft, M. Castro, A. Rowstron, L. Zhou, L. Zhang, and P. Barham, "Vigilante: End-to-end containment of internet worms," in *SOSP*, 2005.

[8] K. Wang, G. Cretu, and S. J. Stolfo, "Anomalous payload-based worm detection and signature generation," in *Proc. of Recent Advances in Intrusion Detection*, 2005.

[9] K. Wang and S. J. Stolfo, "Anomalous payload-based network intrusion detection," in *Proc. of Recent Advances in Intrusion Detection*, 2004.

[10] C. Kruegel, E. Kirda, D. Mutz, W. Robertson, and G. Vigna, "Polymorphic worm detection using structural information of executables," in *RAID*, 2005.

[11] X. Wang, C. Pan, P. Liu, and S. Zhu, "Sigfree: A signature-free buffer overflow attack blocker," in *Proc. of 15th USENIX Security Symposium*, 2006.

[12] N. Weaver, S. Staniford, and V. Paxson, "very fast containment of scanning worms," in *Proceedings of the 13th USENIX Security Symposium*, 2004, pp. 29 – 44.

[13] V. Sekar, Y. Xie, M. K. Reiter, and H. Zhang, "A multi-resolution approach forworm detection and containment," in *DSN '06: Proceedings of the International Conference on Dependable Systems and Networks (DSN'06)*. Washington, DC, USA: IEEE Computer Society, 2006, pp. 189–198.

[14] S. Chen and Y. Tang, "Slowing down Internet worms," in *ICDCS '04: Proceedings of the 24th International Conference on Distributed Computing Systems (ICDCS'04)*. IEEE Computer Society, 2004, pp. 312–319.

[15] S. Sellke, N. B. Shroff, and S. Bagchi, "Modeling and automated containment of worms," in *IEEE DSN*, 2005.

[16] C. Kreibich and J. Crowcroft, "Honeycomb - creating intrusion detection signatures using honeypots," in *Proc. of the Workshop on Hot Topics in Networks (HotNets)*, 2003.

[17] Y. Tang and S. Chen, "Defending against internet worms: A signature-based approach," in *INFOCOM*, 2005.

[18] V. Yegneswaran, J. Giffin, P. Barford, and S. Jha, "An architecture for generating semantic-aware signatures," in *Proc. 14th USENIX Security Symposium*, 2005.

[19] P. Fogla, M. Sharif, R. Perdisci, O. Kolesnikov, and W. Lee, "Polymorphic blending attacks," in *Proc. 15th USENIX Security Symposium*, 2006.

[20] H. J. Wang, C. Guo, D. R. Simon, and A. Zugenmaier, "Shield: Vulnerability-driven network filters for preventing known vulnerability exploits," in *Proceedings of the ACM SIGCOMM Conference*, August 2004.

[21] J. Newsome and D. Song, "Dynamic taint analysis for automatic detection, analysis, and signature generation of exploits on commodity software," in *NDSS*, 2005.

[22] G. E. Suh, J. Lee, and S. Devadas, "Secure program execution via dynamic information flow tracking," in *ASPLOS XI*, 2004.

[23] M. E. Locasto, K. Wang, A. keromytis, and S. J. Stolfo, "Flips: Hybrid adaptive intrusion prevention," in *Proc. of Recent Advances in Intrusion Detection*, 2005.

[24] R. Chinchani and E. V. D. Berg, "A fast static analysis approach to detect exploit code inside network flows," in *RAID*, 2005.

[25] J. Jung, V. Paxson, A. Berger, and H. Balakrishnan, "Fast portscan detection using sequential hypothesis testing," in *Proc. IEEE Symposium on Security and Privacy*, 2004.

[26] G. N. P. R. D. D. Uday Savagaonkar, Ravi Sahita, "An os independent heuristics-based worm-containment system," White paper, 2005.

[27] S. Sellke, "Modeling and automated containment of worms," in *DSN '05: Proceedings of the 2005 International Conference on Dependable Systems and Networks (DSN'05)*. Washington, DC, USA: IEEE Computer Society, 2005, pp. 528–537.

[28] W. R. Group, "Auckland-iv trace archive," 2002. [Online]. Available: http://pma.nlanr.net/Traces/long/auck4.html

[29] 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 Networks (ICON'03)*, October 2003.

[30] P. Porras, L. Briesemeister, K. Skinner, K. Levitt, J. Rowe, and Y.-C. A. Ting, "A hybrid quarantine defense," in *WORM '04: Proceedings of the 2004 ACM workshop on Rapid malcode*. New York, NY, USA: ACM Press, 2004, pp. 73–82.

[31] G. S. Malkin, "Traceroute using an IP option," RFC1393, January 1993.

[32] S. Institute, "Malware faq: What is W32/Blaster worm?" 2003. [Online]. Available: http://www.sans.org/resources/malwarefaq/w32_blasterworm.php

[33] D. Boneh and H. Shacham, "Fast variants of rsa," *CryptoBytes*, vol. 5, no. 1, pp. 1–9, 2002.

[34] A.-R. Sadeghi, M. Selhorst, C. Stüble, C. Wachsmann, M. Winandy, and H. Görtz, "Tcg inside?: a note on tpm specification compliance," in *STC '06: Proceedings of the first ACM workshop on Scalable trusted computing*. New York, NY, USA: ACM Press, 2006, pp. 47–56.