Detection of Hit-list Worms based on Propagation Behavior

February 2008

Nobutaka Kawaguchi

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Background

Today, computer networks have become one of the essential social infrastructures. People enjoy web browsing, shopping, online games, chat and e-mail via the Internet. Governments and companies use the Internet for data exchange, advertisements and e-business. Many organizations introduce LAN and enterprise networks to improve the productivity with small cost.

On the other hand, however, computer networks have suffered significant damages from cyber attacks launched by malicious users. So, it is indispensable to establish countermeasures against the attacks in order to achieve reliable and safe network worlds. While there are various attack techniques such as DDoS, eavesdropping, impersonation, password cracking, computer virus, SPAM mails and spywares [3], this thesis especially focuses on a kind of malicious programs named *Computer Worms*.

Computer worm is a tiny program that propagates itself to many hosts by exploiting their vulnerabilities [4] [5]. Hosts that have been intruded by worms are called *infected hosts*. Infected hosts are basically under the worms controls and used for infecting other hosts and launching various malicious activities [3]. Since the appearance of the first worm, Morris worm [6] in 1988, worms have been one of the most significant threats to the computer networks. Especially, *network worms* that propagate themselves by exploiting vulnerable network services can infect hundreds of thousands hosts automatically without any human interactions.

One of the issues with network worms is how to locate the addresses of vulnerable hosts. Figure.1.1 shows the progress of worm's victim location strategy. In 2001, My SQL Slammer and Codered1, which are the first worms that caused significant damages on the computer network, are born. In this time, worms simply crafted addresses using random generators and attack them. The strategy is called *uniform scan*. Shortly, worms with various more advanced scan strategies that considered the density of vulnerable hosts in address spaces and features of defense systems emerged. Since then, address scans have been the most major victim location strategies and various effective scan techniques have been introduced [7] [8].

At the defense side, most existing detection methods [9] [10] find such worms by detecting address scanners. Since benign hosts rarely conduct address scans, detection of many scanning hosts strongly indicates the existence of worms.

Recently, however, the appearance of new worms that employ different approaches to locate victims hosts is widely discussed among security researchers [11] [12]. Among the future worms, *Hit-list worm* is one of the most important issue. Different form scanning worms, Hit-list worm locates target hosts using a list of victim hosts instead of address scans. Although there is no wild worms that use the complete address lists up to now, some worms have already used a list of hundreds hosts to accelerate the propagation speed at the early stage [13].

There are several reasons that motivate worms authors to create hit-list based worms. Due to the improvement of scan based IDS and population of IPv6, address scans will be less effective. In addition, there have been many useful host databases such as Google, which is used by Sanity worm [14]. Also, as network speed becomes faster, the data size of hit-lists can be ignorable.

Hit-list worm can evade most of existing detection methods since it does not exhibit any aggressive network activities. Although some methods could be applied to detect some kinds of hit-list worms [2] [15], the detection effectiveness is considered to be quite marginal and limited.

Among the hit-list worms, this thesis particularly focuses on worms that attack enterprise networks. The worms are called *Silent worms*. Silent worms have two prominent features. First, Silent worms have a complete address list of victims, and therefore Silent worms never conduct address scans. Second, the number of attack trials per infected host is limited to a few to evade detection systems that focus on the activity of individual host. Due to the features, it will be more difficult to detect Silent worms compared to general hit-list worms. Since the worms can cause significant damages on the organization that owns the attacked network without being noticed by anyone, it is critical to swiftly establish the effective countermeasures to protect the network from such new threats.

So, to detect Silent worms effectively, the thesis proposes a novel approach that detects worms which propagate in an enterprise network by tracking the tree structures composed of anomaly connections [16] [17] [18] [19]. This approach focuses on the essential features of many network worm's propagation behavior, which is that the infection connections can be expressed as tree-like graphs, and realizes faster detection than existing methods. Furthermore, a distributed detection method based on this approach [20] [21] [22] is proposed to realize scalable detection in large enterprise networks.

1.2 Contribution

This thesis discusses how to detect Silent worms quickly in enterprise networks. In general, the many existing network based approaches strongly rely on the worms scans for detection. Thus, they cannot effectively detect Silent worms. In other word, the methods do not sufficiently capture the essential features of propagation behavior common to the most network worms, and which makes difficult for them to detect worms that do not conduct address scans. Another approaches take advantage of packet payloads to detect the existence of worms [23] [24]. They, however, are ineffective against worms with

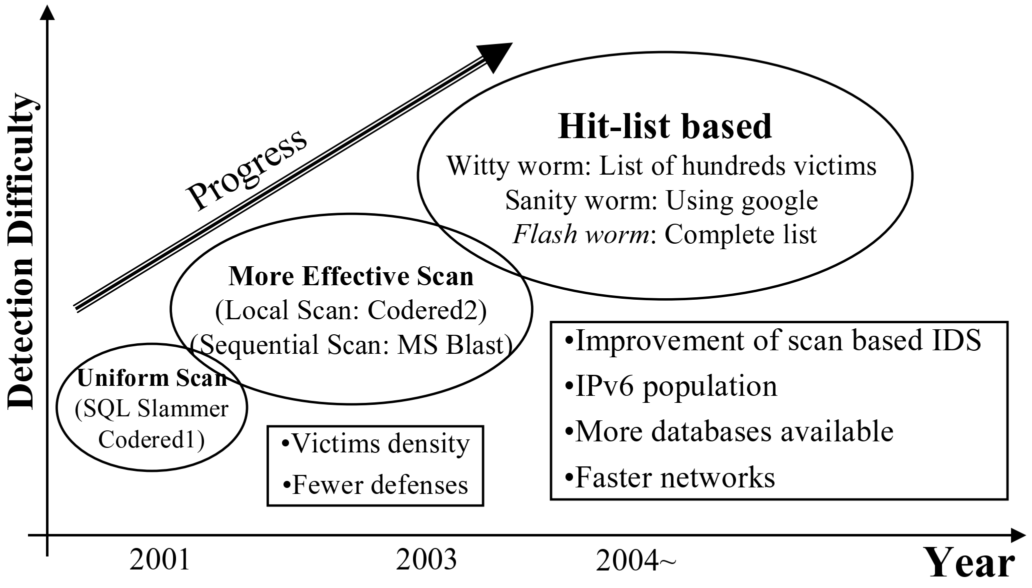


Figure 1.1: Worm's victim location strategy

fully variable payloads and often require to combine with honeypots, which may not be targeted by Silent worms. Payloads are also not the essential features of network worms.

What distinguish worms from other network attacks is that *worms recursively propagate themselves to attack many hosts.* Different from the existing methods, proposed approaches in this thesis focus on the following essential features derived from most network worms propagation behavior.

1. Worms propagation path can be expressed as tree structures with infected hosts as nodes and infection connections as edges

2. Worm often attacks hosts with to which its infected hosts rarely communicate

First, most of network worms have self-propagating abilities. When a host is infected by other host, it soon starts propagation activity that locates and infects other vulnerable hosts. The propagation activity continues until all vulnerable hosts are infected. This results in the appearance of tree structures composed of hosts as nodes and infection connections as edges.

Second, in enterprise networks, most hosts tend to frequently communicate with only a portion of other internal hosts [25] [26]. Thus, since a worm basically does not know to which hosts its infected host has communicated frequently, large portion of infection connections will be established between pairs of hosts that infrequently communicate to each others. Therefore, when worms propagate, tree structures composed of connections between rarely communicating peers will be detected.

Therefore, when worms propagate, tree structures composed of connections that have rarely occurred will emerge, and which can be used for detecting worms. Since Silent worm is a kind of network worms, the discussion here holds for the Silent worm too.

Here, a connection between internal hosts that rarely communicate to each other under normal condition is named as *Anomaly Connection (AC)*, and trees composed of ACs are defined as *Anomaly Connection Tree (AC tree)*.

The main contribution of this thesis is the proposal of two novel detection approaches using the network worms propagation features discussed above.

1. Fast detection of worms based on anomaly connection trees originated from worms propagation.

2. Distributed detection of worms based on the first proposal through the cooperation of several IDSes.

The first proposal detects worms when a detected AC tree exceeds a threshold. In addition, the algorithm also detects an area where many AC trees are detected close together. Even if the size of each tree in the area is under the threshold, the appearance of high-density area of trees will indicate the existence of worms. The area is named as *Virtual AC tree (VAC tree)*, and alerts are raised when the size of the area exceeds a threshold. This approach is called *ACTM (Anomaly Connection Tree Method)*.

In the second approach, AC and VAC tress are detected *in a fully distributed manner*. In a large network, it is difficult for single IDS to monitor and analyze all traffic in the network. The computation and network overheads of the IDS become higher as many hosts join the networks. Moreover, the central IDS can be a single point of failure. To address the problem, in this approach, several IDSes are deployed in a network and cooperate to each others for distributed detection. Each IDS monitors the traffic of a few hosts. Then, through the exchange of the monitored results among IDSes, AC and VAC trees are detected. With small communication overheads among IDSes, this distributed approach realizes the same detection performance as a central approach. The second approach is called *d-ACTM/VT (Distributed ACTM with Virtual AC Tree Detection)*.

Proposed approaches capture the essential features of most network worm's propagation. So, in addition to Silent worms, they can detect the broad range of network worms including scanning worms which propagate themselves into an enterprise network.

1.3 Position of proposed approach

Proposed detection approaches are classified as *Graph based approach*, a kind of network based approaches which detect worms by constructing graphs of connections [2] [15].

Graph based approach will be more effective compared to other approaches such as connection rate and connection failure approaches. Nevertheless, however, the existing methods do not capture the features of worms propagation sufficiently. For example, they depend on the worms payload information [15] [27] or do not consider the anomaly of each connection [2] [28] [29]. As a result, they cannot deal with polymorphic and metamorphic silent worms that change the payloads of attack packets, or worms that spread not so fast and aggressively. As worms propagate slower, the tree structures are deeply hidden behind the background traffic, and therefore, it becomes more difficult to detect the existence by simply aggregating all observed connections to trees.

Thus, this research is the first work that makes possible to effectively detect Silent worms with broad range of propagation speeds and variable payloads through the analysis of connection information.

Table.1.1 shows the position of the proposed methods in graph based approaches. Existing tree based approaches aggregate all observed connections as mentioned above. An existing chain based approach called T.K. algorithm ignores frequent connections and uses only infrequent connections for chain construction. Different from such white list like approaches, d-ACTM/VT [20] and ACTM [18] [19] [17]combine anomaly connections and the other connections named *Normal Connections (NCs)* to detect VAC trees.

The both d-ACTM/VT and GrIDS are distributed IDSes. The difference is that IDSes in d-ACTM/VT dynamically change communication peers according to the topologies of detected AC and VAC trees, while IDSes in GrIDS construct hierarchy structure and each IDS communicates to only predetermined nodes; its parent and children IDSes. Thus, in d-ACTM/VT, network and computation overheads are evenly shared among IDSes and there is no single point of failure. On the other hand, in GrIDS, IDSes at higher levels suffer from considerable overheads and can be a single point of failure. Thus, GrIDS is less scalable than d-ACTM/VT, which is a completely decentralized IDS [30].

Table. 1.2 summarizes the problems with existing approaches and achievements of proposed methods.

1.4 Organization

This thesis is consisted of 5 chapters. Figure 1.2 shows the structure of this thesis.

Chapter 1 showed the background of computer worms, and explained the contribution

Table 1.1: Position of proposed methods in Graph based approaches

		All connections	White list	Combination of AC/NC
Tree based	distributed	GrIDS [2] (Chapter 2)	*no previous work*	**d-ACTM/VT [20] (Chapter 4)**
	central	D.Ellis [28] [29] (Chpater 2)	*no previous work*	**ACTM [18] [17] [19] (Chapter 3)**
Chain based	distributed	*no previous work*	*no previous work*	*no previous work*
	central	*no previous work*	T.K. [15] [27] (Chapter 2)	*no previous work*

and focus of proposed research.

In chapter 2, the taxonomy of computer worms and the defense strategies is presented. Especially, the details of network worms are explained. Then, problems with existing countermeasures against the Silent worm are stated.

Chapter 3 proposes ACTM to detect anomaly connection trees. ACTM not only detects a single tree composed of anomaly connections, but also detects the area where many AC trees densely appear.

Next, chapter 4 proposes d-ACTM/VT to address a problem with ACTM; high computation and network overheads on a central IDS. d-ACTM/VT is a fully decentralized approach, and the computation overhead is evenly shared among IDSes that compose d-ACTM/VT, and therefore is scalable.

Finally, chapter 5 summarizes the achievement of this research and concludes the thesis.

Table 1.2: Problems with existing approaches and achievements of this research

Chapter	Term	Description
Chapter 3	Objective	Fast detection of Silent worms that propagate in enterprise networks
	Problem	Existing approaches are ineffective against worms with changeable payloads and moderate propagation speed
	Proposal	ACTM focuses on the features of worms propagation, and detects worms by detecting tree structures composed of anomaly connections. ACTM also detects area where the trees densely exist
	Achievement	Proposal realizes faster worm detection compared to existing approaches
Chapter 4	Objective	Scalable detection of Silent worms in large networks
	Problem	Existing approaches require a central detection engine or hierarchical structure of IDSes, and therefore are not scalable.
	Proposal	In d-ACTM/VT, each IDS monitor its local target host and exchange information with other IDSes to detect trees in a fully distributed manner
	Achievement	Proposal realizes scalable detection through a distributed manner

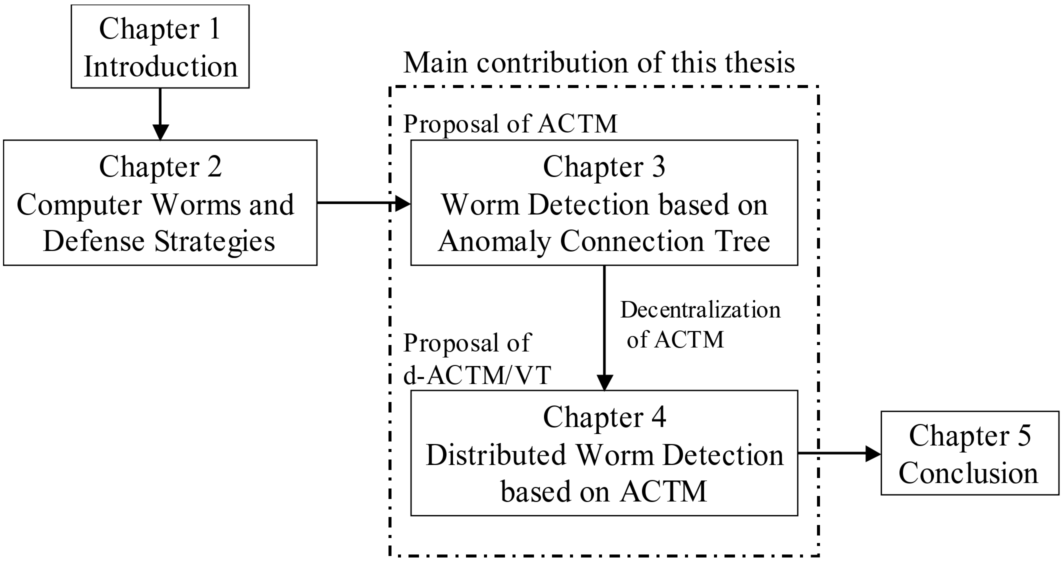


Figure 1.2: Organization of this thesis

Chapter 2

Computer Worms and Defense Strategies

2.1 Introduction

A *computer worm* is a program that self-propagetes across a network exploiting security or policy flaws in widely-used services [4] [5]. Computer worm is different from *computer virus* in that the former is a standalone program that propagates by itself, while the latter attaches to another files for execution.

Since the outbreak of the first worm called Morris worm in 1988 [6], computer worms have been one of the major threats against computer networks. For instance, the series of CodeRed [31] exploit the vulnerability in Microsoft Windows IIS Server and infect the hundreds of thousands hosts in 2001. Moreover, recently, computer worms are used as a propagator of a new threat, Botnet [32], a network consisted of compromised hosts. The botnet is used for launching various attacks such as DDoS, password cracking and spam transmissions. Therefore, understanding the behavior of computer worms and enforcing the effective countermeasures against them are indispensable to accomplish the safe and reliable computer networks.

In this chapter, the taxonomy of computer worms and the defense strategies is shown. Especially, the details of *network worms* are given. Then, problems with existing countermeasures against a kind of hit-list worms named *Silent worm* are stated.

2.2 Classification of Computer Worms

Figure 2.1 shows the classification of computer worms. First of all, computer worms are classified into two categories: *Network worms* and *non-network worms.* Network worms are defined as worms that proactively intrude into hosts by exploiting vulnerabilities of network services performed on the hosts. For example, most of network worms launch buffer flow attacks to get the control of the victim host. On the other hand, *non-network worms* propagate themselves via other means.

Network worms are further classified into two categories based on how to locate vulnerable hosts. *Scanning worms* find hosts with vulnerable network services by scanning network space and ports. There are various scanning techniques [33], which are described in the following sections. Many network worms such as Codered [31], Blster [34], Slammer [35] are classified as this category.

On the other hand, non-scanning worms are network worms that locate vulnerable hosts by some means other than address scans. Among them, worms that utilize a list of addresses of vulnerable hosts are named *Hit-list worms.* Warhol worm is a kind of Hit-list worm [5]. Proposed methods in this thesis attempt to detect a kind of Hit-list worms. Also, worms that obtain the addresses of other hosts from local information of already

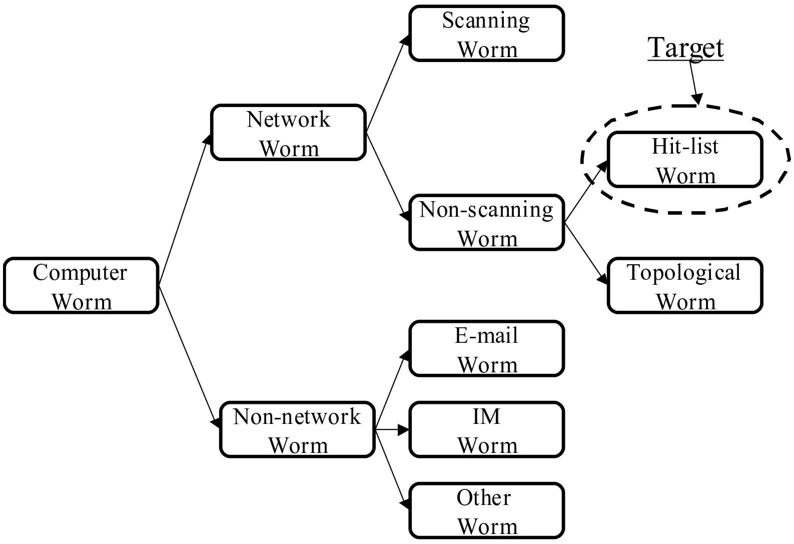


Figure 2.1: Worm Classification

infected hosts are named *topological worms*. Morris worm is a representative one [6].

Non-network worms are roughly classified into *E-mail worms*, *IM(Instant Messanger) worms* and other types of worms. E-mail worms propagate themselves via E-mails [36] [37]. They are usually activated when users open attached files with the mails, or click a link on the body texts [3]. The worms harvest e-mail addresses in the victim hosts and then send themselves to the addresses. Many works have been done in detecting E-mail worms by analyzing traffic [38] [39], tracing mails with attached files [40], inserting fake addresses [41] and so on [42].

IM worms target IM software users, and send messages with malicious links to other users. When a user receives the messages and click links in the message, malicious files are downloaded and executed [43] [44]. Since IM worms send malicious messages to online users directly, they can propagate faster compared to E-mail worms that need to wait users for downloading them from mail servers.

One of the other types of non-network worms is a passive worm. Interestingly, this worm intrudes into a server and then waits for accesses from client hosts [4]. When a client accesses to the server, the worm sends malicious codes to the host and infects it. Since this worm does not open connections to other hosts proactively, it is quite difficult to detect the worm using network-based approaches.

In the later sections, the details of network worms will be shown.

2.3 Scanning Worms

This section describes the scanning strategies of scanning worms and the propagation models.

2.3.1 Scanning Strategies

While address scanning seems quite a simple approach for locating victims, there are various scanning strategies for infecting many hosts quickly and evading detection engines. Existing scanning strategies are classified into six categories as shown in Table 2.1.

Table 2.1: Scanning Strategies

Category	Feature	Example
Uniform Scan	basic strategy	MySQL Slammer [35]
Local Preference Scan	scans local networks preferentially	Codered2 [31]
Sequential Scan	scans address space additively	Blaster [34]
Routing Scan	scans only routable addresses	Routing worm [45]
Cooperative Scan	worms cooperate to scan effectively	[4] [46] [8] [47]
Slowing-down Scan	slow down scan rate for evasion	[48] [7]

Uniform Scan First, *Uniform Scan* is a basic strategy in which a worm randomly selects a target IP address from entire address space (=2^{32} addresses). Most of network worms implement this approach.

Local Preference Scan Worms that employ *Local Preference Scan* select target addresses from the same “/16” or “/8” network more preferentially than entire address space. This strategy is considered to be more effective than uniform scan for the following reasons [33].

1. A network that contains a vulnerable host is likely to have more vulnerable hosts, since hosts in a network can use the same software implementations.

2. Infection connections within internal hosts may not be blocked by firewalls that restricts inbound connections from external networks.

Codered2 [31] implements local address scan module as a part of its scan strategies. This technique implies that merely deploying firewalls at network gateways is not enough for protecting internal hosts once a host in the network is infected, and then it is required to harden LAN level security [49].

Sequential Scan Different from other scanning strategies that randomly select a target address from a given space, *Sequential Scan* worms sequentially scan from a starting IP address selected randomly. Blaster [34] is a typical example. This strategy can certainly scan all addresses in a range of space, but the performance improvement is marginal [33]. Moreover, selection of the starting address influences on the propagation performance. For example, if worms are designed to select starting addresses from their local addresses, worms in the same network may scan the very same addresses for multiple times, and which is quite ineffective.

Routing Scan *Routing Scan* uses BGP routing prefixes to reduce the scanning space. From BGP table, it can be found that only 28.6 % of IPv4 addresses are currently routable. By taking advantage of BGP information and selecting target addresses from the routable space, the routing scan worm can infect the most of vulnerable hosts two times faster compared to simple uniform scanning worm [45].

Cooperative Scan In *Cooperative Scan*, worms instances (worms processes running in infected hosts) communicate to each others in order to effectively scan the network. For example, in *Divide and Conquer Scan*, a newly infected host inherits a half of specific scan space of its infector, and is responsible for scan addresses within the allocated space. Under the assumption where each instance never scan any addresses in its allocated space more than once, the propagation speed continues to be accelerated exponentially to the last [47].

In another strategy called *Importance scan* [8], each worm instance is allocated an address block for scan form a central server, and periodically sends the number of hosts it have infected to the server. Based on the information, the server estimates blocks that likely contain many vulnerable hosts. Then, the server directs infected hosts to scan such blocks intensively. Since the density of vulnerable hosts in an address block is different from each other, concentrating scans on address blocks with high density leads to faster propagation compared to uniform scan.

Slowing-down Scan As more hosts are infected, the amount of scanning traffic increases, and which will cause IDSes to generate alerts. So, to evade the detection engines, *Slowing-down scan worm* restricts its scanning rate according to the number of currently infected hosts. The purpose of the worms is not to infect many hosts, but stealthy obtain the control over a certain number of hosts for malicious activities such as Botnet. Camouflaging worm is a typical one of this class [48]. This worm estimates the ratio of already infected hosts to the number of vulnerable hosts and set the scanning rate to inversely proportional to the ratio.

2.3.2 Propagation Models

Modeling the propagation of scanning worms is beneficial for understanding their effectiveness and planning countermeasures. There are two types of propagation models; *Epidemiological Model* [33] and *Analytical Active Worm Propagation (AAWP) Model* [1]. While the both models can be applied to various situations, here, a simple situation where all suspicious hosts are eventually infected by an uniform address scanning worm is assumed. As to the more complex situations, refer to these papers [50] [31] [51].

Epidemiological Model uses nonlinear differential equation to estimate the number of infected hosts. Assume $I(t)$ denotes the number of infected hosts and N is the total number of suspicious (vulnerable) hosts at a time when worms break out. Ω denotes the address space in the Internet (=2^{32}) and worms scan rate is denoted by η. Now, the number of hosts that are infected at t is expressed as follows.

$$\frac{dI(t)}{dt} = \frac{\eta}{\Omega} I(t)[N - I(t)] \tag{2.1}$$

The solution of 2.1 is

$$I(t) = \frac{I(0)N}{I(0) + [N - I(0)]e^{-\frac{\eta}{\Omega}Nt}} \tag{2.2}$$

where $I(0)$ is the number of initially infected hosts.

On the other hand, AAWP Model is a discrete time model. In this model, $I(t+1)$ can be expressed as

$$I(t+1) = I(t) + [N - I(t)][1 - (1 - \frac{1}{\Omega})^{\eta I(t)}] \tag{2.3}$$

Although the computational cost of AAWP model is higher than Epidemiological Model, AAWP Model is more accurate than Epidemiological Model since this model can take into consideration the case where worm infects the same hosts at the same

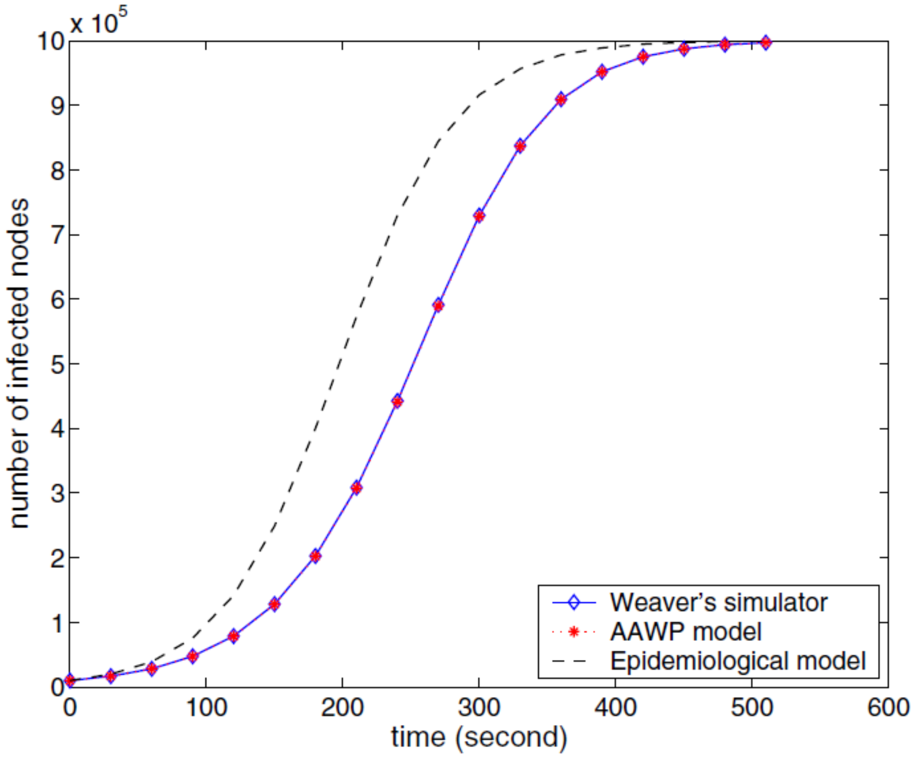


Figure 2.2: Spread of a worm based on Epidemiological and AAWP Models [1]

time, and the period of time taken to complete each infection. On the other hand, since Epidemiological Model does not consider such elements, it may slightly overestimate $I(t)$ of worms that takes a long time to infect each host.

Figure 2.2 shows the spreads of a worm using the models under the condition $I(0) = 10,000$, $\eta = 100/sec$, $N = 1,000,000$. This worm takes 30sec to complete the infection of a vulnerable host. Figure 2.2 also includes a result achieved by a simulator.

At the early stage of propagation, the number of infected hosts increases exponentially ($t < 200$). Then, many hosts are infected at fast near linear speed ($200 < t < 400$). After the worms are saturated in the network ($400 < t$), the propagation speed is down since it becomes difficult to find uninfected hosts. As expected, Epidemiological Model overestimates the number of infected hosts, but the both models accurately estimate when the worm have infected most hosts.

2.4 Non-Scanning Worms

In this section, the classes of non-scanning network worms are introduced. As explained above, most of existing network worms are scanning worms. In the near future, however, the situation can change. For example, the address space of IPv6, which will replace IPv4, is 2^{128}, and 2^{96} times larger than that of IPv4. Then, in the IPv6 Internet, address scanning will be quite ineffective to locate victim hosts. Moreover, the recent advance on the detection engines based on address scan activities makes it difficult for scanning worms to infect many hosts without being detected [12].

So, to handle the situation, worms that locate addresses of victims hosts using some means other than scanning will be popular in the near future [4]. The worms are named as *non-scanning worm.*

The non-scanning worms can be classified into two categories based on how to obtain the addresses as Figure 2.2 shows.

Table 2.2: Non-scanning worms

Category	Feature	Example
Hit-list worm	use hit-list obtained by some means such address scans and databases	Flash worm [11], Witty worm [13] Silent worm [18], Search worm [14]
Topological worm	obtain addresses from local Information of infected hosts in ad-hoc manner	P2P Worm [52] [53] [54] [55] [56], Bluetooth Worm [57] [58], Morris Worm [6],SSH Worm [59], Ad-hoc Worm [60] [61]

Hit-list worms utilize address lists of vulnerable hosts for propagation. They may get the lists from directory servers or address scanning. Among the hit-list worms, this thesis especially focuses on a kind named *Silent worm* [18], and the details will be given in the next section.

On the other hand, topological worms search for local information to discover the local communication topologies and new victims. For example, P2P worms, wireless worms are included in this category.

2.4.1 Hit-list Worms and Silent Worms

Hit-list Worms As explained before, Hit-list worm is a kind of network worm that uses address list of victim hosts. So, how to get the list is the significant issue. The most likely approach is gathering active addresses by long-term stealthy distributed scans [5] [4]. Once a list of hosts running specific applications is generated, attackers can share and re-use the list for the attacks that exploit the application's vulnerabilities. Another approach is to get lists from directory servers such as DNS, Domain Controllers and Search Engines. In fact, Sanity worm released in 2004 takes advantage of Google Search Engine to find web servers that use the vulnerable versions of PHP [14]. As various databases are available and network speed has been faster, it becomes easier to utilize the large size of address lists of vulnerable hosts.

Witty worms is a hit-list worm that attacks a kind of vulnerable IDS in 2004 [13]. This worm uses a hit-list that include several hosts as well as address scans for propagation. S.Staniford and et.al. introduce a theoretical single UDP worm with perfect hit-lists that contain all vulnerable hosts in the Internet. This worm, which is named Flash worm, infects all victims in the Internet in tens of seconds [11]. Similar to divide and conquer worms, each time the worm infects a host, it passes a part of its hit-list to the victim. [62] introduces a worm that randomly generates host names using dictionaries and tries to resolve the names using DNS to find valid IP addresses in use.

As various useful databases are available with low cost and the faster networks eliminate the transmission cost of the large size address list, hit-list worms will be more serious threats in the near future.

Silent Worms This thesis focuses on a kind of hit-list worm named *Silent worm* [18] that attempts to infect all vulnerable hosts in an enterprise network. This worm is modeled as follows.

1. The worm has a perfect hit-list of vulnerable hosts in the target network as Flash worm [11]does.

2. Each instance randomly selects infection targets from uninfected hosts in the hit-list. This means that the worm does not infect already infected hosts more than once.

3. The worm can exploit the vulnerabilities of multiple network services to infect various clients and servers in the network.

4. The number of infection trials of each instance is limited to a few times.

5. The worm infects vulnerable hosts via unicast connections.

6. The worm does not know whether any arbitrary two hosts in a network communicate frequently or not under normal conditions.

A typical vulnerability assumed here is the buffer overflow vulnerability of Microsoft Windows RPC Service exploited by Blaster [34], since both the server and client hosts run this service and therefore many hosts in the network can be infected. (4) is introduced to enable the Silent worms to evade traditional connection rate based detection systems such as Virus Throttle [9]. In addition, Silent worms are assumed not to try to access to the already infected hosts in order to propagate themselves effectively. Using some means such as the division of a hit-list at each infection trial, it is possible to avoid duplicated attacks without any additional communications among worm instance [11] [33].

On the other hand, (6) specifies the limitation of worms knowledge for the target network. As mentioned in [63], hit-list worms are basically defined to have no detailed knowledge other than address lists of vulnerable hosts. This property is important for the proposed method, and therefore the issue will be referred again in later chapters.

Up to now, no wild hit-list worms have complete address lists. In addition, no hit-list worms including theoretical ones limit the number of attack trials of each instance. Thereofore, Silent worm, which is defined in this thesis, exhibits less anomalous network behavior than general hit-list worms. This means that detection of Silent worms can be more difficult compared to other general hit-list worms.

2.4.2 Topological Worms

Topological worms take advantage of local information of infected hosts to locate victims. Morris worm takes advantage of .rhost file in a infected host to locate new targets [6]. SSH worm [59] uses similar approach. On the other hand, P2P worms [52] [53] [54] [55] [56] [64] spread over overlay topologies. Since P2P hosts are connected to many other hosts, P2P worm can spread over the topologies quite fast.

Another types of topological worms try to infect hosts that are physically located at their nearby site. For instance, wireless worms such as bluetooth worms [57] [58] and Ad-hoc worms [60] [61] infect their neighbor hosts via broadcast channels.

Different from the most other network based worms, topological worms may propagate through contacts between frequently communicating peers, and which makes it difficult to detect the worms since their network activities seem to be little anomalous. The detailed discussions will be given in 3.3.7.

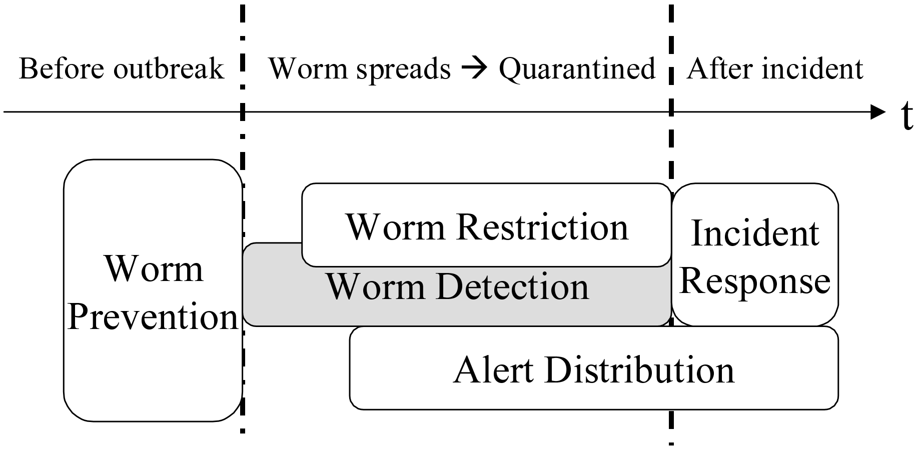


Figure 2.3: Defense Strategies against Network Worms

2.5 Defense Strategies against Network Worms

In the following sections in this chapter, defense strategies to counter the network worms are shown.

Figure 2.3 depicts the overview of defense strategies. The strategies are classified into five categories; *Worm Prevention*, *Worm Detection*, *Worm Restriction*, *Alert Distribution* and *Incident Response*. Among the categories, the major concerns of this thesis are worm detection methods and some restriction approaches that implicitly involve detection phase. Thus, the classes are briefly discussed here, and then discuss the details of detection and restriction in the next section.

Worm Prevention In worm prevention phase, network administrators conduct security analysis and proactive countermeasures for protecting their assets before actual threats appear. Security scanners such as Nessus [65] are used to check and correct the security flows that can be exploited by worms. J.O'Donnel and et.al. propose a software diversity approach for preventing malwares from spreading over the network [66] [67]. Their approach takes advantage of different implementations of software to interrupt the spread of worms that can exploit only a single vulnerability. In host level defense, *Process Address Space Randomization* prevents buffer overflow attacks by randomizing address maps of processes [3]. Also, *Libsafe* replaces the vulnerable functions that can be exploited with secure ones [3].

As a prevention technique against hit-list worms, S.Antonatos and et.al. propose to randomly changes IP addresses of internal hosts over time to decay the address list of the worms [68]. Their approach is based on an idea that more frequently host's addresses are changed, fewer the number of valid addresses in the address list are. Although this approach may be effective against hit-list worms, however, it will be not acceptable to frequently change host's addresses in many organization networks. They also propose to uses a special NAT box to change the external addresses of client hosts frequently. This approach, however, cannot be adapted to server hosts [69].

Worm Detection and Restriction The purpose of worm detection methods is to find the existence of worms as fast as possible by analyzing network traffic and hosts. On the other hand, worm restriction methods try to slow down the activities of network worms by limiting the use of resources such as network bandwidths. Restriction methods are not always activated after worms are detected by some detection methods. Most of the methods automatically limit the activities of a host whenever the host's anomaly level exceeds a threshold, but do not take into account whether the host is actually infected by worms or not. From another viewpoint, it can be said that restriction methods may *implicitly* involve the detection of worms. This is why a part of the area of worm restriction overlaps the area of worm detection in Figure 2.3.

Alert Distribution After IDSes in a domain detect the existence of worms, they may distribute alerts to other domains. Since many worms try to attack the whole Internet, sharing security information across domains make it possible for the administrators to conduct proactive defenses such as port blocking before they receive attacks. There have been many methods for sharing logs and alerts effectively [70] [71] [72] [73] [74] [64].

By combining alerts from individual hosts or domains, fast worm detection in large scale networks such as the Internet is realized with low false positive and negative. [75] proposes an end host based distributed worm detection approach. When a host detects scan activities, it sends a query with *detect=1* to other host that is selected randomly. On receiving the query, if the receiver also detects the scan, it sends a query with *detect=2* to another host, otherwise, it sends a query with *detect=1, non-detect=1.* This process is recursively continued among several hosts, and then if *detect* becomes greater than *non-detect* by a specified value computed with hypothesis testing, worms are detected. This approach is similar to the proposed distributed approach, which will be described in chapter 4, in the points that the both detect worms based on information collected from distributed IDSes.

DOMINO [76] is an architecture for a distributed intrusion detection system that organizes overlay network among heterogeneous networks. The objective is to provide a framework for information sharing aimed at improving intrusion detection capability for all participants. Nodes in DOMINO share their local monitoring results to each others to achieve faster and more accurate detection of scanners, worms and other incidents. [77] takes the similar approach.

Other detection systems that combine alerts from various domains are referred and discussed in 2.6.

Incident Response Incident Response system investigates the damages on the network caused by worms, and locates the intrusion paths. Identifying connections used for worm's propagation is critical to identify the weak points of the network. Yinglian and et.al. propose a forensic approach for identifying such connections by tracing back logs of connection trees with random moon walks [78] [79] [80] [81]. Their approach can identify infection paths with over 90 % accuracy.

2.6 Worm Detection and Restriction

Figure 2.4 shows the classification of detection and restriction approaches. Since some restriction methods involve worm detection phase, Figure 2.4 mixes the both phases. Proposed approach belongs to *Graph based approach*.

First of all, approaches are classified into two categories based on what they monitor; *Host based approach* and *Network based approach*. Host based IDSes are installed in target hosts and monitor the operating systems to detect processes that exhibit anomaly behavior. [82] checks the anomalous file update, creation and deletion. This IDS cooperates with IDSes that monitor other hosts to find files that are updated, created or deleted in multiple hosts. Such files can indicate the existence of worms, since worms instances in different hosts will conduct the same operation on specific files. Similarly, [83] shares the logs of system call sequences with other hosts. Since worms behavior is relatively simpler and more monotonic compared to benign computer programs, activities of worm instances in different hosts will exhibit the almost same system call sequences.

Since host based methods operate inside the target host, they can take advantage of the various detailed information. On the other hand, however, some worms have intelligence to turn off the systems to evade detection [13], and the cost of installing the IDSes to each host in large scale networks is considerably high.

The other approach, network based approaches observes network activities of the target hosts by capturing network traffic. Since network based IDSes can monitor the target hosts from the outside, they are suitable for large-scale networks and are resistant to worms that attempt to kill security programs in infected hosts. The network-based approaches are classified into two categories; *Connection pattern based approach* and *Payload based approach*. Here, A *connection* is defined as a TCP connection or an UDP flow. The pattern of each connection can be expressed as 5 tuples {*source host, destination host, source port, destination port, occurrence time*}. Connection pattern based methods detect worms by focusing on the connection patterns and structures originated from worms propagation such as scanning activities. On the other hand, payload based methods uses payloads of IP packets for detection. Here, many methods employ both the connection patterns and payloads. So, a method, the main concern of which is how to analyze payloads, is classified into payload based approach, and a method that treats payload information as just one of several detection clues is classified into connection pattern based approach.

Connection pattern approach is further classified into three categories; *Connection Failure based approach*, *Connection Rate based approach*, and *Graph based approach*. Connection Failure based approach and Connection Rate based approach focus on the worms address scans that send many packets to unused IP addresses. Connection failure based approaches mainly focus on packets destined to unused addresses. Connection rate based approaches mainly focus on phenomena caused by worm's high rate connection open attempts for many destinations. On the other hand, graph based approach focuses on worm's propagation behaviors across multiple hosts, which construct graph structures composed of infection connections. The proposed methods are categorized in this approach.

In the following sections, the payload approach and these connection pattern approaches are explained.

2.6.1 Payload based approach

Payload based IDSes detect worms by analyzing captured packet contents. The one of the prominent payload based IDSes is Snort [84] that uses signature based approach. A signature represents characteristics of a threat and is written by security researchers and network administrators. In the case of a worm signature, it usually contains a byte sequences and destination ports specific to the worm. Then, by matching the monitored packets with the signatures, worms can be found. This approach accurately detects already known threats with low false positives. D.Moore et.al state that sharing of contents signatures among different domains is more effective in slowing down the worms propagation speed compared to IP address blacklist sharing [85].

The most important issue with the signature-based approaches is how to counter against zero-day worms that exploit unknown vulnerabilities, and therefore there is no

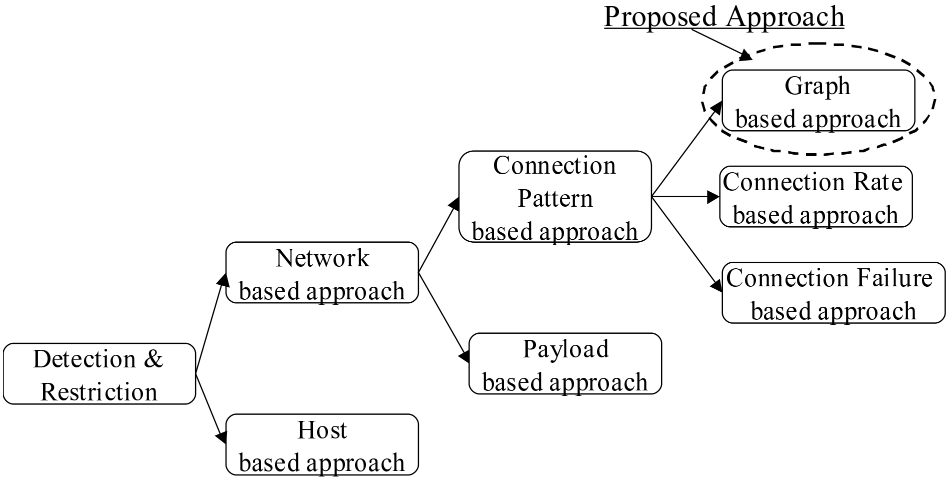


Figure 2.4: Classifiction of Worm detection and restriction

specific signature at their outbreak. To address the issue, there have been some approaches that automatically generate zero-day worms signatures. They focus on the point that since many worms tend to send packets with same payloads to many hosts, the numbers of monitored packets that have same payloads are steeply increased as the worms propagation proceeds. Autograph [24] can generate worms signatures in real-time by finding byte sequences contained in many packets destined to honeypots. [86] [87] use similar approaches. Here, honeypot is a decoy that provides various network services to lure attackers and worms and monitor and record their activities for later analysis. Since honeypot is not used for any legitimate network activities, packets destined to honeypots are supposed to be malicious [88], and the examination of such packets are useful to generate precise signatures. Earlybird [23] [89] counts the number of unique source and destination addresses that send or receive packets with same payloads using Rabing fingerprints, and raises an alert when the count exceeds a threshold. [90] extends Earlybird to consider the communication semantics for packet analysis. WormShiled is a cooperative IDS [91] [92] [93]that uses DHT (Dynamic Hash Table) to count the number of packets that have same payloads, and raises an alert when the count reaches to a threshold.

The advantage of the payload based approaches described above is that they can detect worms independent of the propagation strategies. On the other hand, the most significant issue with payload based approach is how to deal with polymorphic worms and metamorphic worms that can change payloads of attack packets per each infection trial by utilizing different command set, bytes padding and encryption techniques.

Kruegel et al. propose an approach to detect such smart worms by extracting control flow structures of byte sequences in machine language level from packet payloads and finding common flow patterns in variants of an identical worm [94]. Polygraph make signatures that specify the critical patterns of byte sequences which appear in many variants of an identical worm using packet clustering approaches [95]. Hamsa [96] generates signatures composed of multi-sets of tokens while considering the coverage rate in suspicious flows to capture invariants common to polymorphic worms, and outperforms Polygraph in terms of both speed and attack resilence. PADS [97] uses double-honeypots for flow classification and generates position-aware distribution signature that specifies the probability that each byte in an attack sequence takes a certain value. While these approaches can be useful for detecting polymorphic worms, however, they cannot deal with worms transmitted via encryption channels such as SSL/TLS, and attackers still could evade the signatures by injecting noise packets into flow data used for classification [98]. In addition, some of them that use honeypots to collect suspicious flows cannot be applied for the detection of hit-list worms that intuitively attack only the active hosts with hit-lists.

PAYL detects worms by computing the anomaly of packet payloads compared with normal profile. It [99] [100] examines the characteristics of payloads in received packets such as the 1-gram distribution of byte sequences to compute the anomaly score. The anomaly score represents the distance of the received packets from normal profile. Then, if the score is higher than a threshold and the receiver host subsequently also sends out the similar packets to other hosts, PAYL raises an alert. Anagram [101] extends PAYL to counter against *mimicry attack* that inserts normal byte sequences into attack packets to evade anomaly based IDSes.

Helen J. Wang et al. propose an another signature-baed approach called *Vulnerability-Driven Filter* to capture all kinds of worms that exploit a known vulnerability [102]. Different from exploit driven filters (signatures) that describe the exploit codes of the specified worm, the vulnerability driven filter describes a known vulnerability (e.g. insertion of data exceeding l at a certain session on the service s leads to buffer overflow attacks). In this approach, a detection module, which is called *Shield*, is installed in each host and intercepts traffic above the transport layer. Then, it tracks the status of sessions between its host and the peer from captured network data, and identifies vulnerability exploit attempts using the vulnerability filters. BrowserShiled extends this method to deal with dynamic contents such as dynamic HTML in web pages [103]. IntroVirt is an another type of vulnerability driven filter approach [104]. Different from Shiled that analyzes network traffic, this analyzes the execution and state of the target system based on virtual machine monitoring to find malicious codes that exploit vulnerabilities described in filters. The drawbacks of these approaches are that they need human operators to write filters manually and cannot deal with unkown vulnerabilities.

On the other hand, Vigilante [105] makes it possible to generate signatures for unknown vulnerabilities automatically. In this system, honeypots generates SCA (Self-certifying alert) that proves the vulnerabilities of a specific software by examining data in incoming messages. The SCA is broadcasted to many domains. After verifying the SCA with virtual machines, the receiver makes filters by performing dynamic data and control flow analysis of the execution path the worm follows to exploit the vulnerability described in the SCA. COVERS [106] generates signatures by identifying packet payloads that cause memory errors while considering the particular message fields where the attack codes are embed. ARBOR [107] takes a similar approach based on program behavior, These vulnerability-based approaches can generate signatures for unknown vulnerabilities with high accuracy and low false positive. On the other hand, they require host-based IDSes to analyze attacks and various knowledge such as protocol/application specifications.

2.6.2 Connection Failure based approach

As explained in previous sections, many network worms do address scans to find victim hosts. During this process, many packets are destined to unused address space and unused ports. Connection failure based approach takes advantage of the feature to detect the existence of scanners. There have been many methods that use the feature [47] [108] [109] [110] [111] [10] [112] [113] [114] [115] [116].

As a most simple approach, [108] [117] judge a host as a scanner if it has sent a number of packets to unused addresses in a period of time. [116] focuses on broadcasted ARP requests for unused addresses to detect scanners in LAN. One of the most prominent methods in detecting a scanner is Sequential Hypothesis Testing based method [10]. To judge whether a host r is a scanner or not, this method takes the sequence of the latest N outcomes of connection attempt by r as input. The outcome is either of *success* or *failure*. Now, St denotes the sub sequence from $N-t+1$ th outcome to N th (the newest) outcome. In St, the number of failure outcomes is t_f and the number of success is $t-t_f$. The anomaly score of r with t sequential outcomes is computed as

$$A_t = \frac{(1-S_s)^{t_f} * S_s^{t-t_f}}{(1-B_s)^{t_f} * B_s^{t-t_f}} \tag{2.4}$$

where S_s and B_s are prior probabilities where a connection opened by a scanner and benign host are successful respectively. Then, if there exists $t(1 \leq t \leq N)$ that satisfies $A_t \geq TH_s$, r is judged as a scanner. This method can detect a scanner before it has sent out ten scan packets with low false positive. This approach can detect scanners faster than [108] [117].

[47] proposes the victim based approach that counts the number of suspicious hosts that send packets for unused addresses and detects worms when the number exceeds a border, and illustrates that this approach outperforms an approach that simply counts the number of suspicious packets. [112] raises an alert of worms when a host receives scan packets from other hosts, and then subsequently starts address scans. This approach focuses on *the chain of the scanners*, and then distinguishes the worms propagation activities from mere scanners. [109] [110] [111] monitor TCP Reset packets and ICMP Host Unreachable Messages to estimate the number of scanners in the Internet. [114] [115] use large-scale network telescopes and detect worms when the number of hosts that send packets to unused space is increased at the rate that matches the propagation model of network worms described in 2.3.2. For the analysis, this method gathers packet data from various domains using distributed IDSes. In [113], different domains share the lists of source hosts that have recently sent packet to their domain's unused addresses to effectively conduct countermeasures such as packet filtering at firewalls.

2.6.3 Connection Rate based approach

Connection rate based approach takes the advantage of the worm's propagation feature that the high rate address scanning sends many packets with a specific destination port to the large number of destinations. Generally, connection rate based approaches have advantages over failure connection based approaches since they do not need to judge whether a connection is destined to an unused address or valid address. Currently, many organizations filter out ICMP Host Unreachable packets and TCP Reset packets from their internal hosts for security reasons, and therefore it becomes not easy to verify which packets are destined to unused addresses.

Connection Rate based approaches are further classified into three categories as follows.

1. Number of connections based approach
2. Balance between addresses and ports based approach
3. Number of communication with unfamiliar hosts based approach

The first approach focuses on the number of connections in a time of period. The second approach focuses on the balance between addresses and ports of observed packets, which can be changed by worm's high volume traffic. The third approach focuses on the connections between hosts that rarely communicate under normal condition.

Number of connections based approach Virus throttle [9] limits the number of new connections that each host can opens in a period of time. Virus throttle is installed in each host and monitors its outgoing connection attempts. Virus throttle has two components called *working set* and *queue*. Working set maintains a list of recently accessed hosts. When a host tries to send a packet to initiate a connection (e.g. TCP SYN packet), Virus throttle interferes the packet. Then, if the destination host of the packet is not listed in the working set, the packet is stacked on the queue. Thus, the queue periodically sends out the saved packet (e.g. 1 packet /sec) to restrict the connection attempt rate. Finally, if the size of the queue exceeds a predefined limit, Virus throttle regards the host as a scanner or an infected host, and all subsequent outgoing packets are to be dropped for a certain period of time.

[118] proposes *credit* based approach. In this approach, each host initially has 10 credits. Then, as the host tries to open a new connection, one credit is consumed. If the connection attempt is success, two credits are added. Then, if all credits are consumed, the host cannot open any connections for a certain period of time. Also, to prevent worms from using many credits that a host has saved for a long time, if a host has more than 10 credits, a third of them is dropped for each second. Different from Virus throttle, this approach allows benign hosts to open connections with high rate. Since some applications may causes such burst connection open attempts (e.g access to a web page that contains many external resources), this credit based approach can be more accurate than Virus throttle, which simply blocks applications with high connection rate regardless of the success ratio.

Similarly, [119] detects the existence of worms by detecting the change point of host's connection degree. [120] [121] deploy the rate limiting filter at edge routers and the Internet backbone, and show that filtering at the backbone and edge router levels are more effective than host level filtering. [122] uses inbound connection throttling where inbound connections for a network are rate limited by ingress routers as well as outbound throttling to further slow down the propagation. [123] also restricts the connection from the Internet to enterprise networks to limit the number of infected hosts under a given acceptable value. [124] discusses a strategy to deploy a limited number of throttling filters to a given number of networks while considering the importance of each network. [125] [126] limit the number of unique destination hosts to which each host can access in a period. Since worms need to access to many hosts for propagation, the approaches are quite effective. [127] focuses on the interval between when a host receives packets for a port and when the host subsequently sends out a first packet for the same port. Since worms try to spread fast, the interval will be very short. Thus, by forwarding suspicious packets to virtual machines and monitoring the timing and data size of inbound and outbound packets, worms can be detected.

Balance between addresses and ports based approach [128] observes backbone traffic, and detects worms when the ratio of the number of unique destination addresses of observed packets to the number of unique ports is significantly changed. Since worms traffic is destined to a specific port to which the vulnerable services listen, the ratio will be increased steeply due to the worms high scan rate. [129] [130] compute entropies of source and destination addresses, and source and destination ports. As worm spreads, entropies of source addresses and destination ports are decreased, and destination addresses and source ports are increased. The methods use the features for detection. [131] detects worms by identifying suspicious destination ports and then counting the number of unique destination hosts of outbound connections for suspicious ports from internal hosts to external hosts.

Number of communications with unfamiliar hosts based approach [132] monitors communications between internal hosts in the network under normal condition for a certain time and makes a normal profile that includes pairs of frequently communicating hosts. Then, connections that are not included in the profile are regarded as probably anomalous and throttled at switches. This approach focuses on the point that a worm probably do not know to which hosts the infected hosts frequently have communicated when it starts propagation. With a similar approach, [133] monitors traffic at a gateway and specifies the frequent communication external hosts of each internal host, and then restricts the accesses to unfamiliar external hosts to prevent worms propagation from internal hosts to external hosts. On the other hand, [134] [135] restricts inbound connections from unfamiliar hosts in the Internet to rarely accessed servers in local networks to prevent worms propagation from external to internal hosts.

2.6.4 Graph based approach

Graph based approach detect connection graphs that are likely to be constructed by worms propagation. Different from other network-based approaches, this approach focuses on the structures composed by hosts and connections between them rather than the statistics derived from traffic of a single host or networks. Since this approach does not depend on worm's aggressive scanning behaviors, it could be applied to detect non-scanning worms.

Currently, there are two types of graph based approach, *Tree based approach* [2] [28] [29], and *Chain based approach* [15] [27].

Tree based approach Stuart Staniford and et.al. firstly propose a tree based IDS [2] named GrIDS. GrIDS monitors network activities in an enterprise network and tracks tree structures caused by worm's propagation. Figure 2.5 show the overview. In this figure, a tree structure rooted at host A is constructed. An edge and small rectangle indicate a connection and host respectively. To detect tree structures of connections, GrIDS aggregates connections that occur within a certain time to others. Then, when the size of detected tree exceeds a threshold, GrIDS raises an alert. The size of a tree is the number of hosts included in the tree. The size of a tree in Figure 2.5 is 10.

GrIDS is composed of several IDSes that constructs hierarchy structures. Each IDS is responsible for monitoring hosts in its local network. Then, IDSes at higher side aggregates information from IDSes at lower side. At the left side of Figure 2.5, there are 3 IDSes and each of them monitors hosts in an inner dotted rectangle. Also, there is one higer level IDS, which is described as an outer rectangle. Three lower side IDSes monitors host A-B, C-G and host H-J respectively and send the monitored results to the higher-level IDS. A higher side IDS can estimate the tree size using information from lower side IDSes. The right side of Figure 2.5 depicts what the higer-level IDS can see. The higer-level IDS sees only the summaries of the monitoring results of lower-level IDSes, and the details of tree structures are hidden. This abstraction reduces the overloads of higher level IDSes.

GrIDS is the first work that reflects the propagation behaviors of network worms, that is, tree structures composed of infection connections are certainly constructed. GrIDS will detect most of aggressive scanning worms since the high rate infection connections for many hosts quickly make large trees.

On the other hand, however, since GrIDS just uses the tree size for detection, it cannot effectively detect relatively slow non-scanning worms, whose interval of continuous infection connections is almost same to that of legitimate connections opened by benign hosts. In addition, since GrIDS is a distributed IDS, and consists of several IDSes deployed in a hierarchical manner, computation network overheads at higher level IDSes can be high. Also, higher-level IDSes can be a single point of failure.

Dan Ellis and et.al. implement an IDS similar to GrIDS [28] [29] with a single server. They show this approach is effective against high connection rate scanning and hit-list worms.

Chain based approach Thomas Toth and et.al. firstly propose a chain based approach [15], so called T.K. algorithm. Figure 2.6 shows an example. This method aggregates an inbound connection at t_{in} and outbound connection at t_{out} of a host only when $t_{out} > t_{in}$ and $t_{out} - t_{in} < threshold$. Next, this method counts the number of connections in the chain which are *similar* to each others. For instance, connections that have same payloads and destination ports are regarded as similar connections. Then, the number of similar connections is used as an anomaly score for threshold based detection. [27]

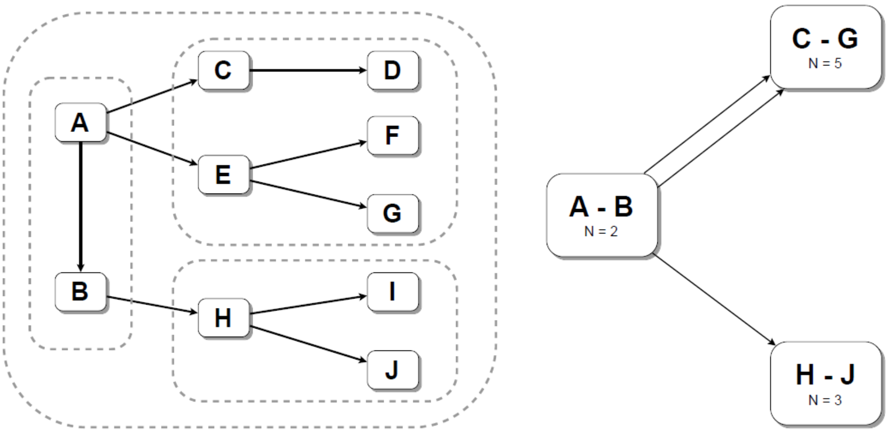


Figure 2.5: Overview of GrIDS [2]

improves the performance of [15] by filtering connections using a while-list and some algorithm optimizations. The white-list includes *well-known* hosts and services in the network. Pairs of hosts and ports, to which many connections are destined frequently, are regarded as *well-known*. Then, connections for hosts and ports included in the white-list are excluded from detection targets. As a result, the computation cost is significantly reduced compared to [15].

2.7 Problem Statement and position of proposed approach

2.7.1 Problem Statement

Now, the problem in detecting Silent worms using the existing network based approaches is stated. Table 2.3 shows the detection performance of these approaches against Silent worms. As noted, the existing approaches are unable to effectively detect Silent worms that change attack payloads and spread not so fast. The reasons are as follows.

First, payload based approach is completely ineffective if the worm is either polymorphic or metamorphic. Connection Failure based approach is completely ineffective since no infection connections are destined to unused addresses.

Connection Rate based approach is marginally effective only when the Silent worms

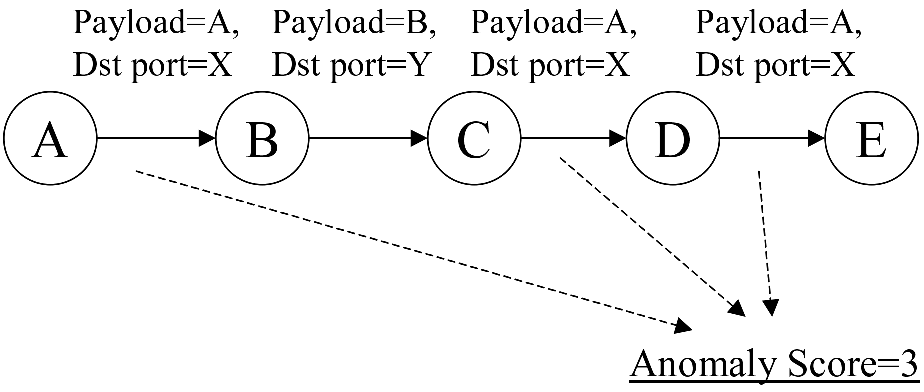


Figure 2.6: Overview of Chain Approach

spread fast. Since the Silent worms limit the number of infection trials per each instance to a few, host based throttling [9] [132] approaches are ineffective. Connection rate based approach is basically dependent on statistics such as the number of connections in a period, and do not take the underlying propagation behavior of worms into consideration.

Tree based approach could be the most promising approach to detect hit-list worms since it does not focus on the scanning behaviors of worms, but tracks the propagation behaviors of worms. GrIDS [2] and its successors [28] [29] simply aggregate connections that are made closely in time, however, and they do not take the *anomaly of each connection* into consideration during for detection process. As a result, they cannot deal with worms that propagate not so fast (moderate speed) since the tree structures are deeply hidden behind the background traffic. Also GrIDS suffers from high computation and network overhead since many information are aggregated to IDSes at higher-level.

Another graph based approach, chain based approach (T.K. algorithm) [15] [27] detect worms by computing anomaly score based on the number of similar connections. Since payloads matching basically calculate the similarity between connections, it cannot deal with polymorphic or metamorphic worms effectively. Moreover, the shapes of connection chains do not fully reflect the worms propagation tree structures. Although [27] states that the chain approach is still able to deal with polymorphic worms by calculating the similarity of connections based on the destination port numbers only, the detection performance will be significantly decreased.

In summary, features that most existing approaches leverage for detection such as *high connection rate*, *many connection failures* and *variable attack payloads* are not the essential features of worms propagation behavior. Using hit-list, worms no longer need to open connections to random addresses with so high rate, and their payloads can be changed using encryption channels.

In a word, the essential features of the most network worm's propagation behavior are that the worms infection connections are rarely occurring connections, and construct tree like structures. Although some graph based approaches focus on a part of the essentials, their way are still insufficient (they do not consider the anomaly of each connection).

As contrasted with the existing approaches, proposals focus on the essential features of worms propagation well, and therefore they will be still effective against Silent worms with variable payloads and moderate propagation speed

Table 2.3: Detection performance against Silent worms

Approach Type	Performance
Payload	completely ineffective agianst worms with variable payloads
Connection Failure	completely ineffective
Connection Rate	completely ineffective againt worms with moderate propagation speed
Tree ([2] [28] [29])	ineffective against worms with moderate propagation speed
Chain([15] [27])	ineffective against worms with variable payloads
Proposed([18] [20])	still effective against worms with variable payloads and moderate propagation speed

2.7.2 Position of proposed Approach

For the detection of the Silent worms more effectively, *Anomaly Connection Tree based Approaches* are proposed.

Different from the existing methods, proposed approaches in this thesis focus on the following essential features of the most network worms propagation behavior.

1. Worms propagation path can be expressed as tree structures with infected hosts as nodes and infection connections as edges
2. Each worm instance is likely to attack hosts with which its infected hosts rarely communicate under normal conditions

First, most of network worms have self-propagating abilities. When a host is infected by other host, it soon starts propagation activity that locates and infects other vulnerable hosts. The propagation activity continues until all vulnerable hosts are infected. This results in the appearance of tree structures composed of hosts as nodes and infection connections as edges.

Second, in enterprise networks, most hosts tend to frequently communicate with only a portion of other hosts [25] [26]. Thus, since a worm basically does not know to which hosts its infected host has communicated frequently, large portion of infection connections will be established between pairs of hosts that infrequently communicate to each others. Therefore, when worms propagate, tree structures composed of connections between rarely communicating peers will be detected.

Therefore, when worms propagate in a network, tree structures composed of connections that have rarely occurred will emerge, and which can be used for detecting worms. Since Silent worm is a kind of network worms, the discussion here holds for the Silent worm too.

The proposed approaches detect worms by tracking trees composed of connections that are rarely made under normal condition. Since this approach does not utilize payload information for detection, it is still effective against polymorphic and metamorphic worms. Discussions about exceptional worms that do not have such features are given in 3.3.7.

In chapter 3, *Anomaly Connection Tree Method (ACTM)* that implements this approach is proposed. ACTM not only detects a single tree composed of anomaly connections, but also takes a *distance* between multiple trees into consideration using a novel approach to detect Virtual AC trees.

Next, in chapter 4, a distributed approach of ACTM named *d-ACTM/VT* is proposed to achieve scalable detection. Different from GrIDS, d-ACTM/VT is a fully decentralized approach, and therefore there is no single point of failure and the computation and network overheads are evenly dispersed among IDSes that compose d-ACTM/VT.

Chapter 3

Worm Detection based on Anomaly Connection Tree

3.1 Introduction

In this chapter, Anomaly Connection Tree Method (ACTM) is proposed to detect zero-day Silent worms that propagate in an enterprise network. ACTM takes advantage of two features of the propagation activities of most worms including Silent worms. First is that worms propagation activities can be expressed as tree-like structures composed of infected hosts as nodes and infection connections as edges. Second is that the worms do not consider which hosts its infected host communicates to frequently when selecting infection targets. Then, ACTM detects the existence of the worms by detecting tree structures composed of Anomaly Connections (AC). Different from GrIDS [2], ACTM considers the anomaly of each connection, and therefore is still effective against worms whose infection connections interval is equal to the average interval of legitimate connections opened by internal host.

The following sections are organized as follows. In section 3.2, ACTM is proposed. In section 3.3, the performance of ACTM is evaluated. Finally, section 3.4 summarizes this chapter.

3.2 Anomaly Connection Tree Method

This section presents a novel approach named Anomaly Connection Tree Method. First, the connection model used in this chapter is described. Then, the detailed algorithm of ACTM is introduced.

3.2.1 Connection Model

ACTM will detect the worms that attempt to infect all vulnerable hosts in a network such as intranet and LANs. A communication between two internal hosts in the network is named an IC (Internal Connection). The ICs includes TCP connections, UDP flows between the internal hosts. An IDS that executes ACTM for worm detection accumulates IC logs from several packet capture devices in the network and analyzes them. Each IC is identified by the tuple of source host and destination host. A source host is a host that opens a connection and a destination host is a communication peer of the connection.

Here, there are two ways to classify ICs. One way classifies ICs based on the activities that open the ICs. An IC opened by a legitimate network activity is classified into a LC (Legitimate Connection) and an IC opened by a worm's infection activity is classified into a WC (Worm Connection).

The second way classifies ICs based on the occurrence frequency of the ICs. An IC, the occurrence probability of which in normal operations exceeds a threshold is classified into a NC (Normal Connection), and an IC, the occurrence probability of which is under the threshold is classified into an AC (Anomaly Connection).

ACTM recognizes which ICs are ACs or NCs by analyzing the frequencies of occurrence of ICs in connection logs for a certain period of time. Notice, on the other hand, ACTM cannot *directly* recognize which ICs are LCs or WCs since the detection targets of ACTM are zero-day Silent worms and any contents signature is not provided to identify WCs.

As mentioned above, ACTM recognizes ICs by source and destination hosts. Two ICs that share a host (e.g. an IC between host A and B, and an IC between host A and C) are different from each other. Also, two ICs with different directions are recognized as different connections even if they share two end hosts (eg. an IC from A to B and an IC from B to A).

3.2.2 Overview of Detection Algorithm

ACTM detects the existence of worms by detecting ACs and tree structures composed of the ACs as edges and infected hosts as nodes. ACTM uses following two features of worm propagation activities for the detection.

1. The worm copies itself onto the infected hosts repeatedly and the newly infected hosts will try to infect other hosts. Then, the worm's infection activities can be expressed as a tree structure composed of WCs as edges.

2. When selecting infection targets, the worm does not consider to which hosts its infected host has communicated frequently under normal conditions

On the other hand, benign hosts tend to frequently communicate to a small portion of all hosts [25] [26]. For example, 80% of all connections opened by a host can be destined to the 20% of all hosts in the network.

ACTM logs LCs via packet capturing for a certain period of time under the condition that there is no worm in the network. Then ACTM classifies LCs, the occurrence rate of which exceeds an threshold, into NCs. And the other ICs are classified into ACs. As benign hosts tend to communicate to a small portion of all hosts with high frequently, many of LCs are classified into NCs. On the other hand, worms propagate themselves without considering the frequent communication hosts of its infected host. As a result, many of WCs will be classified into ACs.

Next, ACTM concatenates monitored ACs to detect anomaly connection trees (AC trees). As mentioned above, ACTM cannot directly recognize WCs. But, since many of WCs will be classified as ACs, ACTM can detect worms propagation by detecting large

AC trees. Figure 3.1 shows examples. In this figure, there are some ACs, NCs, and two AC trees. One tree is composed of host {A, B, D, E, H}, and the other is composed of host {C, F, G, I}. The probability that a LC is classified into an AC is smaller than the probability that a WC is classified into an AC. Thus, when worms propagate, large AC trees are more likely to be detected compared to normal conditions. ACTM therefore estimates the existence of worms when a detected AC tree exceeds a threshold. Here, the tree size means the number of hosts that compose the tree. In Figure 3.1, the size of the two trees are 5 and 4 respectively.

Since a few WCs are still detected as NCs, however, a tree constructed by worm propagation activities can be detected as not a single AC tree but as a set of some trees divided by the NCs. In this case, some relatively large AC trees tend to be detected within a short distance from each others. ACTM therefore aggregates AC trees within a short distance from each others and detects a set of the trees as a single VAC tree (Virtual AC tree). Then, similar to the case of AC trees, when a VAC tree exceeds a threshold, ACTM detects the existence of worms. Details of the algorithm such as a definition of *distance* will be given in the later sections.

ACTM has two phases, the learning phase and the detection phase. In the learning phase, ACTM observes the network for a certain period of time and determines which ICs should be classified into NCs or ACs and the thresholds for the detection of AC and VAC trees. It is assumed that there is no worm in this phase. In the detection phase, ACTM detects AC, VAC trees in real-time and detects the existence of worms using thresholds determined at the learning phase.

In the following sections, the IC classification algorithm, AC,VAC tree detection algorithm and the learning and detection phases are explained.

3.2.3 IC Classification Algorithm

Here, how to classify ICs into ACs and NCs at the learning phase is explained. As mentioned before, in this phase, ACTM obtains the logs of observed ICs in the network for a period of time. Then, for each host (e.g. host X), ACTM creates a list named Clist, which lists the number of ICs the host X opens for each destination host. The destination hosts include all internal hosts and are sorted according to the number of ICs the host X has opened to the hosts. Figure 3.2 shows the Clist of the host X.

Here, the top FR ($0 \leq FR \leq 1$) destination hosts in the Clist are named as *Fhosts*. ACTM classifies the ICs opened by the host X to Fhosts as NCs and the other ICs as ACs. Here, let's denote *CR* as the ratio of the number of NCs that a host has opened at the learning phase to the total number of ICs that the host has opened. Since worms select infection targets randomly from the hit-list, the probability that a WC is an AC

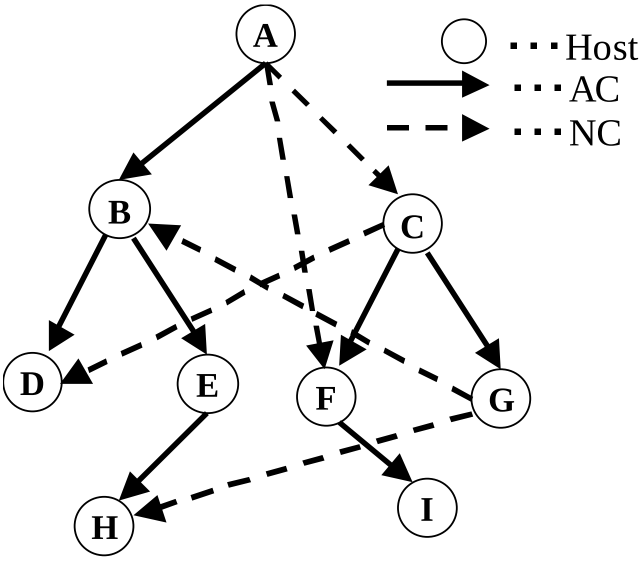


Figure 3.1: Example of AC Trees

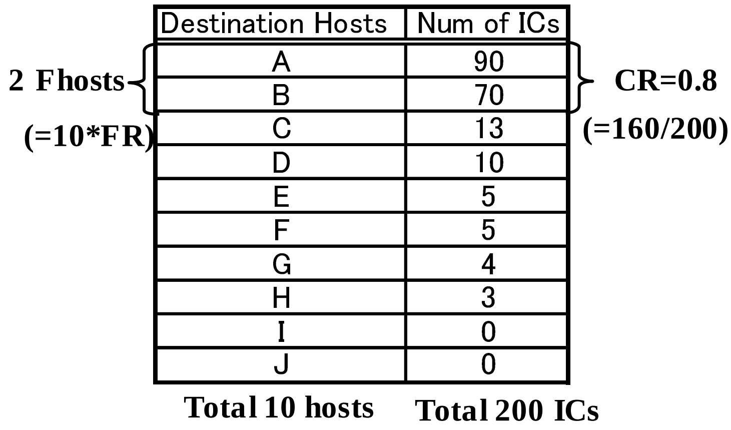
can be expressed as $1.0 - FR$. On the other hand, the probability that a LC is an AC can be expressed as $1.0 - CR$.

In Figure 3.2, total number of destination hosts is 10 and FR is set to 0.2. So, the X's Fhosts are two hosts; host A and B. Therefore, ICs destined to the host A or B are classified as NCs and the other ICs are classified as ACs. CR of the host X is 0.8 $(= (90 + 70)/200)$. As a result, 80% of WCs and 20% of LCs are classified as ACs.

3.2.4 AC Tree Detection Algorithm

By concatenating newly detected ACs with already detected AC trees, ACTM keeps searching for larger AC trees. Figure 3.3 shows the detection process of AC trees by concatenating new ACs. In this figure, an AC is represented as an arrow with a source host as an origin point and a destination host as an end point. The detection time of an AC is represented as the time that the end point of corresponding arrow indicates. An AC with source host X, destination host Y and detection time Z is denoted as $AC^{Z}_{X,Y}$. For example, an AC shown in Figure.3.3(a) is represented as $AC^{T_1}_{A,B}$.

Here, assume ACTM detects a new AC X. If the origin point of X is the origin or end point of an AC Y which was detected within a period of T_{limit} (*Connection Limit Time*), ACTM concatenates X with Y. When X can be concatenated with more than one ACs,



Destination Hosts	Num of ICs
A	90
B	70
C	13
D	10
E	5
F	5
G	4
H	3
I	0
J	0

2 Fhosts (=10*FR)

CR=0.8 (=160/200)

Total 10 hosts

Total 200 ICs

Figure 3.2: Clist for Host X

X is concatenated with the AC that belongs to the largest trees. Figure 3.3 shows the detection of AC trees when $T_2 - T_1 \leq T_{limit} < T_3 - T_1$. In Figure 3.3(b), $AC^{T_2}_{B,C}$, $AC^{T_2}_{B,D}$ are concatenated with $AC^{T_1}_{A,B}$ since the detection interval $T_2 - T_1$ is smaller than T_{limit}. Then, the size of the AC tree TR1 increases by 2.

On the other hand, when there is no AC that satisfies the concatenation condition, a newly detected AC becomes the first AC of a new AC tree. For example, in Figure 3.3(C), $AC^{T3}_{A,G}$ is not concatenated with $AC^{T_1}_{A,B}$ and becomes the first AC of a new tree TR2.

3.2.5 VAC Tree Detection Algorithm

When FR is set to small enough, the probability that a WC is an AC ($= 1.0 - FR$) is higher than the probability that a WC is a NC ($= FR$). But a few WCs are still classified as NCs, and such WCs can divide the worms propagation trees into some AC trees. In this case, as Figure 3.4 shows, instead of a single large AC tree, some moderate size AC trees divided by the NCs are detected. In Figure 3.4, TR2 and TR3 are separated from TR1 by two NCs and totally 3 AC trees are detected. So, the WCs classified as NCs have bad effect on the performance of ACTM that detects worms based on the size of AC trees. To reduce the effect, ACTM employs another algorithm, *VAC Tree Detection Algorithm*.

First, the distance between two trees (TR1 and TR2, TR1 and TR3) is defined as the length of shortest sequence of NCs that concatenates two trees. When a tree caused by worms propagation is divided into two trees by NC(s), the probability that the distance between the trees is d is $FR^d * (1.0 - FR)$. Thus, when FR=0.2, the probability that

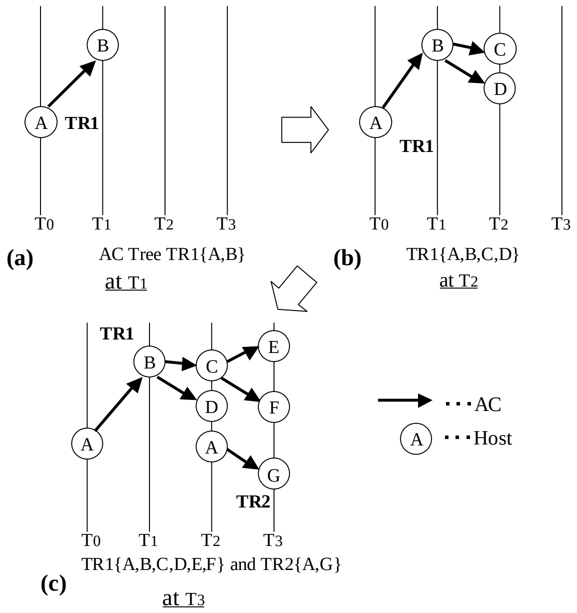


Figure 3.3: Detection of AC Trees

$d \geq 2$ is 0.04. Then, when an AC tree caused by worm's propagation is divided by NCs, the probability that the divided trees still exist close to each other is high. Therefore, by aggregating trees within a certain distance from an arbitrary tree into one tree, the most part of worm's infection tree structure divided by NC(s) can be recovered.

ACTM aggregates an arbitrary tree (a center tree) and its neighbor trees as a set named VAC tree. Then, if the size of VAC tree exceeds a threshold, the existence of the worms is detected. In Figure 3.4, a VAC tree VTR1 composed of TR1 as a center tree, TR2 and TR3 as neighbor trees is detected.

This algorithm is applied for all AC trees. This means, for each AC tree as a center tree, the size of the corresponding VAC tree is calculated. As a result, when there are N AC trees, the sizes of N VAC trees are calculated.

Here, V_d, V_n are defined as parameters to determine which AC trees are aggregated as neighbor trees. V_d is a maximum distance of neighbor trees from the center tree (In Figure 3.4, V_d=1). V_n is the maximum number of the neighbor trees (In Figure 3.4, V_n=2). When there are S_d neighbor trees NTR_i within the distance V_d from the center tree CTR $(1 \leq i \leq S_d, sizeof(NTR_i) \geq sizeof(NTR_{i+1}))$, the size of VAC tree VTR is computed

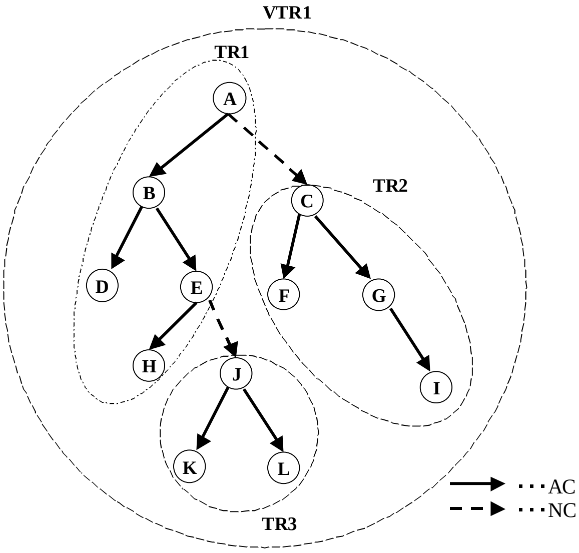


Figure 3.4: Detection of VAC Tree

as

$$sizeof(VTR) = sizeof(CTR) + \sum_{i=1}^{V_n} sizeof(NTR_i) \tag{3.1}$$

In the case $S_d < V_n$, $V_n = S_d$.

As Equation(3.1) shows, of S_d neighbor trees, ACTM aggregates only top V_n trees for the VAC tree. The reason is that, since many small AC trees are detected even if there is no worm, the aggregation of small size trees are ineffective for the detection of worms propagation trees.

In this algorithm, a NC $NC1$ concatenates two AC trees TR1 and TR2 when each tree has an AC that shares the origin point or end point with NC1 and the detection interval time between NC1 and the AC is within T_{limit}(connection limit time). When the distance between TR1 and TR2 is longer than one, the trees are concatenated by the sequence of more than one NC. In this case, the concatenation condition between any adjacent NCs is same as mentioned above.

3.2.6 The Learning and Detection Phase

In the learning phase, ACTM observes the network for a certain time period and generates IC logs. Then, ACTM determines which ICs should be classified into ACs or NCs. Next, AC and VAC trees are constructed from the logs, and the largest size of the AC tree and the VAC tree are set to the thresholds TH_{AC} and TH_{VAC} respectively. It is assumed that there is no worm during this phase.

In the detection phase, ACTM detects AC and VAC trees in real-time. Then, ACTM detects the existence of worms when a tree that exceeds the threshold, is detected.

3.3 Evaluation

In this section, the computer simulation to evaluate the effectiveness of ACTM is conducted.

3.3.1 Simulation Condition

In this simulation, an intranet where the all-vulnerable internal hosts are targeted by Silent worms is assumed. Firewalls do not block connections between the internal hosts, and then each host can communicate to any host freely. In the detection phase, one Silent worm instance infects one internal host by some means and starts to infect all hosts. The worm uses TCP connections as the infection connections.

Table 3.1 shows the parameters used in this simulation. Each host opens LCs to the other hosts. The interval between the two continuous LCs opened by a host (legitimate connection interval) follows the exponential distribution and the average is set to 10TU. Here, TU denotes the time unit. As to the connection model of internal hosts, FR and CR are set to 0.2 and 0.8 respectively for all hosts as default values. The Fhosts of each host are selected from all hosts randomly. The number of infection trials of each infected host is 2 at most as a default value. For the detection side, this condition is quite strict. The infection interval is the average interval between the start times of 2 continuous WCs of a worm copy. The interval between when an host is infected and when the host starts infection activity is also the same value. The behavior model of benign hosts and infected hosts are derived from Xie's model of worms propagation in enterprise networks [78] [80].

In this simulation, the learning phase is set to 10000 TU. The TH_{AC} and TH_{VAC} are set to the size of the largest AC and VAC trees observed in this phase. In the detection phase, after 1000 TU passes, one worm copy infects a host and starts propagation. The number of infected hosts is measured when worms are detected by ACTM.

The evaluation criteria are as follows.

1. The number of infected hosts before worms are detected.

2. Comparison to the another detection methods.

Table 3.1: Simulation Parameters

# of hosts	1000
FR / CR	0.2 / 0.8
Legistimate connection interval	10 TU
# of infection trial per instance	2
Infection interval	1-20TU
V_d	1
V_n	2

3. The effect of connection limit time on the detection performance.
4. The effect of false positive rate on the number of infected hosts.
5. The effect of the bias of communications on the number of infected hosts.

For comparison, Virus throttle [9] and AC Counting Method are employed. Virus throttle detects hosts that try to open connections for many hosts in a short interval. Every time a host tries to open an IC, the connection initiation packet (e.g. SYN packet) is pushed into a queue for the host. In every fixed interval, a packet is popped from the queue in FIFO and sent to the destination. Virus throttle detect the worms when a queue overflows. In this simulation, the method is modified to recognize which ICs are ACs, and push only the AC initiation packets into the queues. As a result, the detection performance of the modified one is improved compared to that of original one.

AC Counting Method counts the number of ACs opened by all internal hosts for a certain period of time, and detects the worms when the number exceeds a threshold. Similar to ACTM, the largest value in the learning phase is used in the detection phase as the threshold. Note, AC Counting Method is also an original method as well as ACTM.

3.3.2 Comparison of the number of infected hosts before detection

Figure 3.5 shows the number of infected hosts before ACTM and AC Counting Method detect the worms. In this figure, the connection limit time is optimized for each infection interval. When infection interval is shorter than 14TU, ACTM can detect the worms faster than AC Counting Method. This indicates the use of worm's infection tree structures

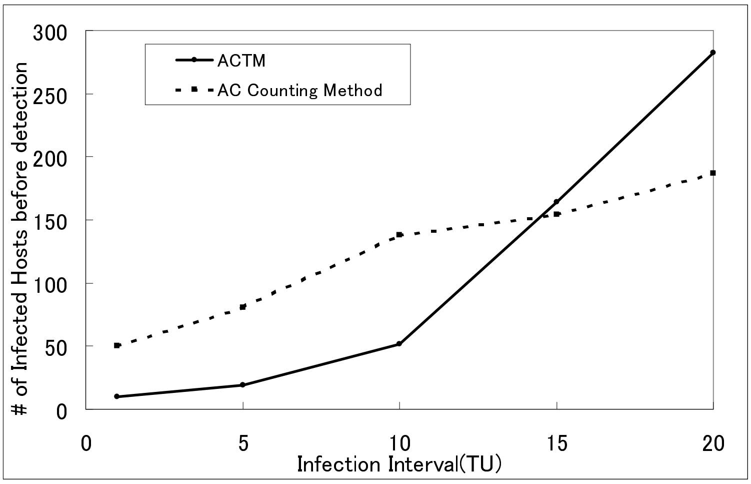


Figure 3.5: Comparison of Infected Hosts before Detection

contributes to the fast detection. For example, when the infection interval is equal or shorter than the legitimate connection interval (10TU), ACTM can detect worms before 5% of all hosts are infected. On the other hand, when the infection interval is longer than 14TU, AC Counting Method becomes faster than ACTM. This is because, as the infection interval is longer, TH_{AC}, TH_{VAC} becomes larger As a result, the probability that NCs interfere the growth an AC tree before it exceeds a threshold gets higher and it becomes difficult to recover the most part of infection tree by VAC tree detection algorithm. Generally, however, the infection intervals of most existing worms are several times shorter than the intervals of LCs. Thus, it can be said that that the propagation speed of a worm whose infection interval is same as LC interval is enough moderate or even slow. Therefore, ACTM is more effective than AC Counting Method against most worms.

Also, the number of infected hosts before Virus throttle detects the worms with 1TU and 10 TU infection intervals are 519 and 720 respectively. So, the method is much slower than ACTM and AC Counting Method. The reason is that since the number of infection trials per instance is limited to a few times, the probability that the queue of a host overflows is quite small.

Figure 3.6 shows the number of infected hosts when the number of infection trials of each instance is varied from 2 to 8. The number of infected hosts with ACTM is almost constant. On the other hand, the numbers of infected hosts with AC Counting Method and Virus throttle increase as the number of trials decreases. This indicates that ACTM

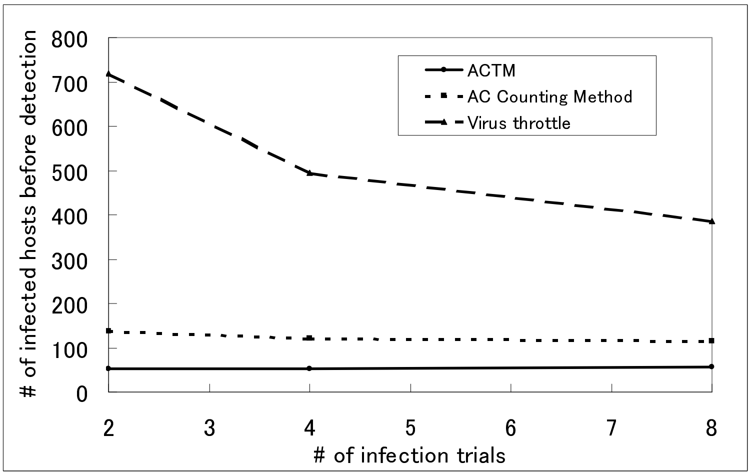


Figure 3.6: The Effect of the Number of Infection Trials with Infection Interval=10TU

is more effective against worms with various number of infection trials compared to other detection methods.

3.3.3 The effect of connection limit time and detection parameters

Figure 3.7 shows the effect of the connection limit time on the number of infected hosts before detection. When the connection limit time is slightly larger than or equal to the infection interval, the number of infected host is minimized. This is because if the connection limit time is smaller than the infection interval, ACTM cannot concatenate two continuous WCs, and if connection limit time is too larger than the infection interval, TH_{AC} and TH_{VAC} becomes significantly large and detection is delayed.

Table 3.2 shows TH_{AC}, TH_{VAC} and the ratio of the number of detections by VAC Trees to the number of simulation trials. As the infection interval increases, the probability that ACTM detects worms with VAC trees rather than AC trees becomes higher. The reason is that as the infection interval increases, TH_{AC} is increased, and therefore the probability that AC trees are separated by WCs classified as NCs before they exceed TH_{AC} gets higher.

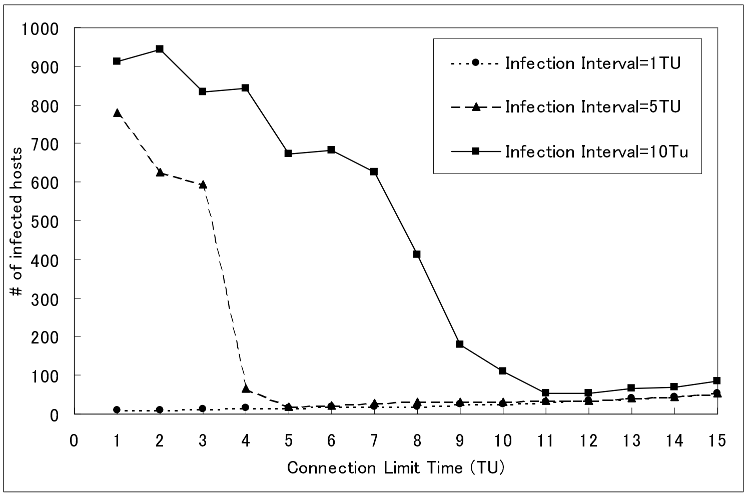


Figure 3.7: The Effect of the Connection Limit Time

Table 3.2: Detection Parameters

Infection Interval(TU)	1TU	5TU	10TU
TH_{AC}	7	11	21
TH_{VAC}	8	16	31
Ratio of the # of detections by VAC Tree	15%	35%	50%

3.3.4 The effect of false positive rate on the number of infected hosts

Figure 3.8 shows the effect of difference of TH_{AC} and TH_{VAC} values from *standard thresholds* on the false positive rate and the number of infected hosts when the infection interval is 10TU.

Here, difference from standard thresholds means the amounts of changes of TH_{AC} and TH_{VAC} values from the thresholds shown in Table 3.2 (TH_{AC}=21, TH_{VAC}=31). For example, in the case where the difference is "-10", TH_{AC} and TH_{VAC} are set to 11 and 21 respectively, and in the case where the difference is "+15", TH_{AC} and TH_{VAC} are set to 36 and 46 respectively.

False positive rate (FPR) is the ratio of the number of VAC/AC trees that exceed the thresholds to the number of all trees under the condition where there is no worm in the network. Here, in this simulation, about 10 new trees are detected per 1TU on average when there is no worm.

As Figure 3.8 shows, FPR is about 0.0 when thresholds exceeds the standard thresholds. When a threshold is set to the standard thresholds, 0-1 false positive is generated per each 10000 TU.

To detect worms faster, set the thresholds to smaller values. If thresholds are smaller than the standard thresholds by 5, ACTM detect worms when 33 hosts are infected. In this case, FPR is about $1.0 * 10^{-4}$ and a false positive alert is raised per 500 TU.

3.3.5 The effect of the bias of communications on the number of infected hosts

As each internal host tends to frequently communicate to a smaller portion of all hosts, ACTM can detect worms faster since the difference between normal network activities and worms infection activities becomes bigger.

The *Bias Score* is defined as the ratio of the number of 80% of all internal hosts to the number of top destination hosts for 80% of all ICs opened by a host. As a host tends to communicate frequently to a smaller portion of all hosts, the Bias Score of the host becomes larger. For example, with FR=0.2 and CR=0.8, the Bias Score is $4.0(= (1000 * 0.8)/(1000 * 0.2))$. Also, in the case where a host tends to communicate with all hosts evenly, the Bias Score is 1.0. For simplicity, here it is assumed that the Bias Scores of all hosts take same values.

Table 3.3 shows the number of infected hosts with various Bias Scores and infection intervals. As the Bias Score is larger, ACTM can detect the worms faster. On the other hand, as the score is smaller, more hosts are infected before detection. For example, when

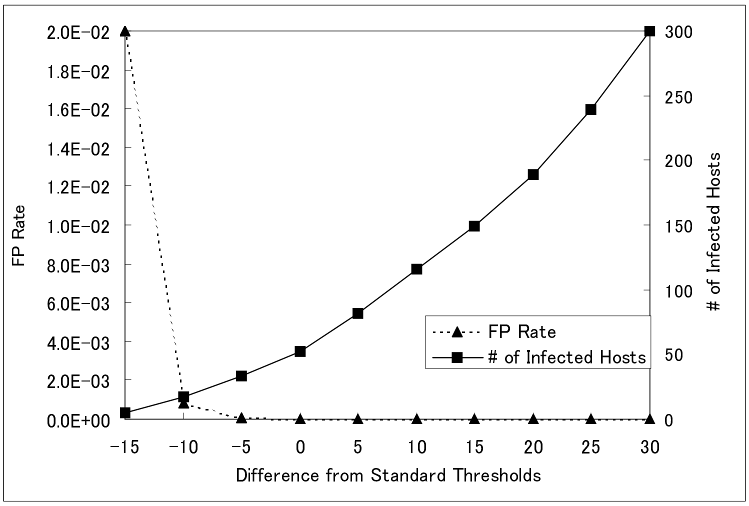


Figure 3.8: The Effect of False Positive Rate with Infection Interval=10TU

Table 3.3: Effect of the Bias of Communications

Infection Interval (TU)	1TU	5TU	10TU
Bias Score=8.0	7	14	29
Bias Score=4.0	11	19	52
Bias Score=2.0	13	62	133
Bias Score=1.0	14	288	986

the Bias Score is 1.0 and the infection interval is 10TU, most hosts (986 hosts) are infected before detection.

Here, note that since GrIDS [2] does not consider the bias of connections for tree detection, its detection performance is same as the performance of ACTM with Bias Score=1.0. So, the results in Table 3.3 shows GrIDS is completely ineffective against worms with Infection Interval=10TU. In most networks, the Bias Score will be larger than 1.0. Therefore, it is concluded that ACTM is much effective than GrIDS against Silent worms.

3.3.6 Performance for other network worms

Since ACTM takes advantages trees of anomaly connections, which are the essential features of worms propagation, this method is also effective against most other network worms other than hit-list worms that propagate themselves recursively and do not care about the communication patterns of victim hosts.

For example, assume there is a scanning worm that scans the same entire B class address space, and 1000 vulnerable hosts are allocated in the space. In this case, per 65 scans, a active host is hit on average. Since connections destined to unused addresses are intuitively considered to be anomaly connections, an AC tree with 65 nodes is detected before the worm infects any hosts. Here, Table. 3.2 shows that TH_{AC} is quite smaller than 65 generally. Thus, ACTM can detect scanning worms before any additional hosts are infected.

So, ACTM is effective against most network worms. There is, however, one exception; topological worm. Since each topological worm instance attacks hosts by examining its infected host's address lists, the infection connections are likely to be classified as NCs if the address list includes only the frequently communicating hosts. For example, if the worm uses ARP caches, which contains addresses of recently communicating hosts in a network, the worm may attack only the frequently communicating hosts via NCs. As a result, AC/VAC trees exceeding the thresholds are unlikely to be detected. The more discussions will be given in the next section.

3.3.7 Limitation of ACTM

Although ACTM can detect the broad rage of network worms under most conditions, there are two types of network worms that can evade the detection as follows.

1. Worms with quite low propagation speed
2. Worms that conduct propagation via NCs

First, as Figure 3.5 shows, the detection performance of ACTM becomes worse as worms propagate slower. For example, when the infection interval is 40TU, which is the four times of average LC interval, it is almost impossible to detect the existence before all hosts are infected. On the other hand, however, as worms propagate slower, the chance that effective signatures or software patches are provided by venders before many hosts are infected becomes higher. Thus, in terms of results, as worms propagate slower, the number of eventually infected hosts can be smaller although ACTM cannot detect their existence. Thus, ACTM is useful to induce worms authors to design slow propagation worms, which are allowed to infect only the small number of hosts before effective countermeasures are performed.

Second, worms that have ability to propagate themselves via NCs can evade ACTM since no large AC/VAC trees are constructed. An example is a topological worm. Since the worm finds victims from address lists of already infected hosts, most of infection connections can be classified as NCs. On the other hand, however, there have been works that detect topological worms through *deception*. The proposed methods insert *dummy addresses* into address lists of potentially vulnerable internal hosts [136] [137]. Dummy addresses are addresses that are not allocated to any legitimate active hosts. Thus, no connections to the dummy addresses are made under normal conditions. On the other hand, since it is difficult to worms to identify which addresses are dummy addresses, connections to the dummy addresses will be made when the topological worm propagates, and accesses to address books containing dummy addresses. Therefore the occurrence of connections to dummy addresses indicates the existence of topological worms.

Incidentally, if all active addresses in a network instead of dummy addresses are inserted into each host's address list or each host have all internal addresses in advance, ACTM can detect topological worms with the same performance as topological worms.

Thus, although there are some types of network worms that can evade ACTM, they can be defeated by other means, and their performance is limited. Thus, ACTM is effective in significantly narrowing the design space of network worms that can infect many hosts.

In addition, in the following network environments, the performance of ACTM can be significantly degraded.

1. Most internal hosts *evenly* communicate to many hosts

2. P2P applications that construct tree-like connection topologies are used

As to the first case, if hosts evenly communicate to many other hosts, connections with the most combinations of source and destination hosts are classified as NCs. As a result, worms propagations unlikely construct large AC/VAC trees. As stated in [25] [26], however, most internal hosts communicate to only a few percent of other hosts in the same network. Thus, ACTM is considered to be enough effective in most networks.

As to the second case, some types of P2P applications such as file sharing software construct tree-like connection topologies like worms, and which may cause many false alerts. By filtering out the traffic caused by P2P applications, the bad effect can be reduced. In addition, in reality, many organizations prohibit the use of such P2P applications in enterprise networks for the security reasons.

3.4 Summary

To detect Silent worms in enterprise networks, this chapter has proposed Anomaly Connection Tree Method (ACTM). ACTM uses two features of most worms. First is that the worm's propagation behavior constructs tree-like structures. Second is that the worm's selection of infection target does not consider which hosts a host communicates to frequently. Then, by detecting trees composed of anomaly connections, ACTM detects the existence of such worms. Through computer simulation experiments, it has been shown that ACTM can detect Silent worms before 5% of all vulnerable hosts are infected under the condition where the infection interval is equal or shorter than the legitimate connection interval.

Chapter 4

Distributed Worm Detection based on ACTM

4.1 Introduction

This chaper proposes distributed detection of Silent worms based on ACTM [16] [18].

One of the significant issues with ACTM is that it requires a single detection engine that has global knowledge of the network such as all connection logs to detect AC tree structures. As the network size increases, it becomes more difficult to accumulate and analyze all network logs in the network to a single detection engine. Thus, more scalable approach is required to realize fast and efficient detection in large enterprise networks.

To address the issues, following two methods are proposed.

1. d-ACTM (Distributed ACTM) [21] [22]
2. d-ACTM/VT (d-ACTM with Virtual AC Tree Detection) [20]

First, d-ACTM detects AC trees in a distributed manner. d-ACTM is a distributed IDS composed of several IDSes named LACDs (Local AC Detectors). Each LACD monitors network activity of its target host and classifies the connections into AC/NC categories. Then, LACDs exchange their local monitoring results to detect AC trees in a distributed manner. A root node of an AC tree estimates the tree size based on the information received from its children nodes.

Next, to detect VAC trees as well as AC trees, d-ACTM/VT is proposed. d-ACTM/VT is an extension of d-ACTM, and aggregates trees divided by NCs into VAC trees in a distributed manner. In d-ACTM/VT, for tree aggregation, information messages about the detected AC trees are sent forward from IDSes monitoring the root hosts of the trees to IDSes monitoring the root hosts of AC trees at upward side. Here, ”upward side” represents the relative position of an AC tree that is located at a root direction of other AC tree.

d-ACTM/VT makes possible to detect Silent worms in a distributed manner with the same detection speed as ACTM.

Another graph based approach, GrIDS [2] uses hierarchical IDSes to disperse the overheads into several IDSes, but nodes at higher-levels still suffer high computation and network overheads and can be a single point of failure [30].

The following sections are organized as follows. Section 4.2, presents the algorithm of d-ACTM. Section 4.3 states the problem with d-ACTM and then proposes d-ACTM/VT. In section 4.4, the performance of d-ACTM/VT is evaluated. Finally, section 4.5 summarizes this chapter.

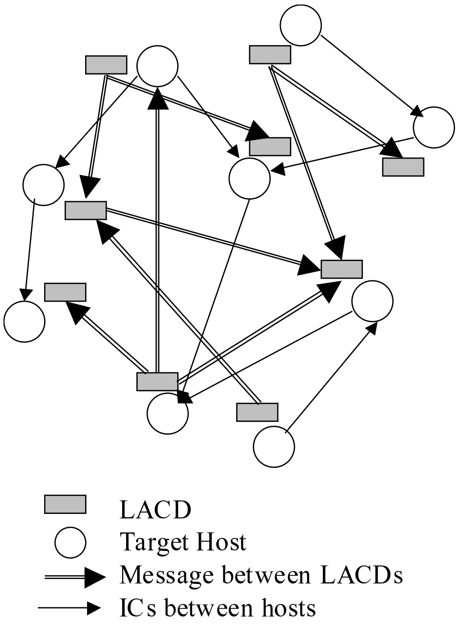


Figure 4.1: Overview of d-ACTM

4.2 d-ACTM

4.2.1 Overview of d-ACTM

d-ACTM detects the existence of worms through the distributed detection of tree structures composed of the ACs as edges and infected hosts as nodes. Figure.4.1 shows the overview of d-ACTM. In d-ACTM, Local Anomaly Connection Detectors (LACDs) monitor connection of target hosts and cooperatively detect the AC tree structures in a distributed manner. Here, the *target host* means the host, which a LACD is responsible to monitor. In this paper, it is assumed that each LACD monitors only one target host.

As well as ACTM, d-ACTM takes advantage of the following two features of worm propagation activities for detection.

1. The worm copies itself onto the infected hosts repeatedly and the duplicated worms will try to infect other hosts. Then, the worm infection activities can be expressed as a tree structure composed of infected hosts as nodes and WCs as edges.

2. When selecting infection targets, the worm does not care about which hosts and how

frequently its infected host has communicated to when the host was not infected and under normal operation.

As contrasted with worms, on the other hand, benign hosts in a network tend to communicate to a small portion of all the internal hosts frequently [25] [26]. For example, 80% of all connections opened by a host could be destined to 20% of all hosts in the enterprise network.

To identify ACs and NCs, each LACD captures IC logs of its target host by some means. Then, in each period of a certain time, outbound IC logs captured for a certain period are classified into neither of ACs or NCs. The length of the period specifies the scope of logs analyzed at the same time.

LACD classifies ICs based on the occurrence rate of each IC. ICs with high occurrence rate are classified into NCs and the other ICs are classified into ACs. As benign hosts tend to communicate to a small portion of all internal hosts with high frequency, many of LCs are classified into NCs. On the other hand, many WCs will be classified into ACs because the worm propagates without caring about to which hosts its infected hosts usually frequently communicate.

Next, by exchanging some types of messages, LACDs concatenate ACs and detect anomaly connection trees (AC trees) in a distributed manner. LACDs cannot directly identify WCs. But, since most of WCs will be ACs, LACDs can detect worms infection trees by cooperatively detecting large AC trees.

When the size of an AC tree exceeds TH_{ac}, LACD raises an alert of worm detection. The alert is verified by the *network manager*, a system that investigates the network for alert verification. Then, using the verification result, each LACD automatically adjusts its TH_{ac} in order to realize the desired interval of false alert issuance.

4.2.2 LACD

Figure.4.2 shows the structure of LACD. LACD has an ability to monitor and capture the inbound and outbound ICs of its target host. Each LACD just identifies ICs from the traffic and does not need to analyze the details of packets payloads. Thus, LACD can be a network based IDS as well as a host based IDS. In addition, in this paper LACDs are assumed to be secure and not compromised by any worms or other types of attacks.

LACDs have the following three main roles.

1. Each LACD classifies outbound ICs of its target host into either of ACs or NCs.

2. LACDs communicate with each others and detect AC trees cooperatively.

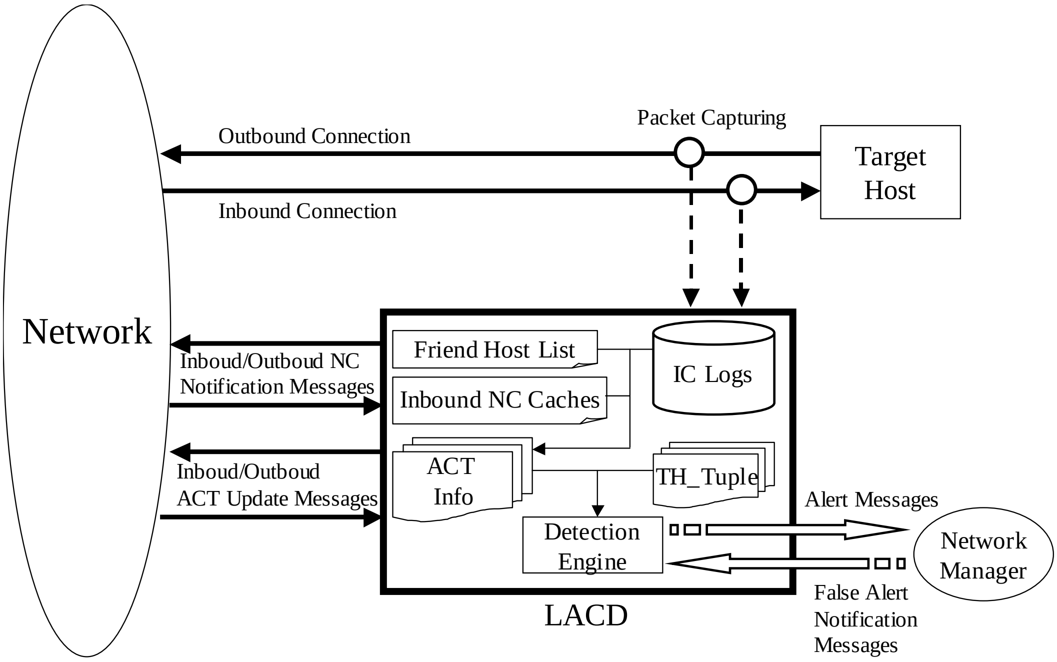


Figure 4.2: Structure of LACD

3. LACDs automatically adjust TH_{ac} to realize a desirable balance between detection speed and false alert rate.

To carry out the roles, LACDs maintain the following data.

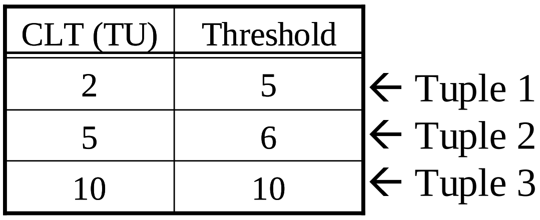
1. IC Logs: is the outbound IC logs of the target host.

2. Friend Host List(FHL): is the list of hosts, outbound ICs destined for which are classified into NCs.

3. Inbound NC Cache: preserve the hosts, inbound ICs from which are classified into NCs. Each cache consists of the IP address of the host and the expiration time.

4. ACT_Info: maintains the information about an AC tree in which the target host is included. Each ACT_Info corresponds to one AC tree.

5. TH_Tuple: consists of Connection Limit Time(CLT) and TH_{ac}. Figure.4.3 shows an example. Here, TU is a time unit used throughout this paper. CLT is a parameter to



CLT (TU)	Threshold	
2	5	← Tuple 1
5	6	← Tuple 2
10	10	← Tuple 3

Figure 4.3: TH_Tuples

concatenate ACs in order to detect AC trees. When the size of an AC tree detected with a CLT exceeds TH_{ac} corresponding to the CLT, LACDs raise an alert. LACD may have multiple TH_Tuples for the fast detection of worms with various infection speeds. In Figure.4.3, there are 3 tuples. For example, if the size of a tree detected with CLT=2 exceeds 5, the LACD raises an alert.

To detect worms in a distributed manner, each LACD exchanges two types of messages with one another.

1. NC Notification Message(NNM)
2. ACT Update Message(AUM)

LACD sends a NNM to inform other LACDs that an IC from its target host is a NC. Also, LACD sends an AUM to inform the growth of a detected AC tree. In d-ACTM/VT, some other types messages are added, but the details are given in the next section.

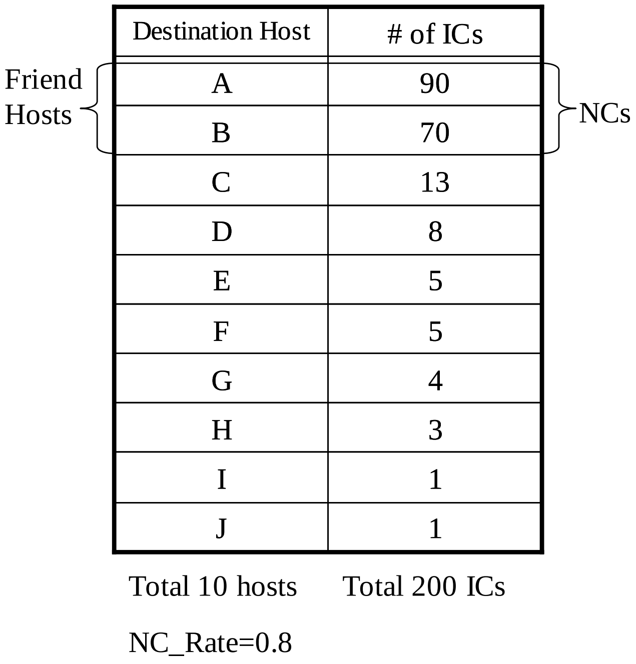
When a LACD detects the worms, it sends an alert message to the network manager. The network manager conducts detailed investigation of the network and hosts to verify the received alerts. The investigation takes some amount of time. Then the network manager returns a false alert notification message to the LACD if the alert is found to be a false alert.

In the following sections, the details of IC classification, message transmission are given.

4.2.3 IC Classification

In each classification interval time(CIT), each LACD analyzes its target host's outbound IC logs monitored for the past classification window time(CWT) in order to classify ICs into either of NCs or ACs. For example, assume CIT=I_c, CWT=W_c and LACD starts operation at t_0. At $t_0 + I_c$, logs from $t_0 + I_c - W_c$ to $t_0 + I_c$ are analyzed, then at $t_0 + 2I_c$, logs from $t_0 + 2I_c - W_c$ to $T_0 + 2I_c$ are analyzed, and so on.

In the analysis, as shown in Figure.4.4, the tuples of hosts and the number of outbound ICs its target host has sent to the hosts are aligned according to the number of ICs in descending order.



Destination Host	# of ICs
A	90
B	70
C	13
D	8
E	5
F	5
G	4
H	3
I	1
J	1

Friend Hosts
NCs
Total 10 hosts
Total 200 ICs
NC_Rate=0.8

Figure 4.4: NCs and Friend Hosts

Next, LACD adds top hosts in the sorted list to a list named FHL (Friend Host List) until the total number of ICs destined to the hosts in FHL becomes greater or equals to *NC_Rate*total number of outbound ICs*. Then ICs destined to the hosts listed in FHL are classified as NCs and other ICs are classified as ACs. In Figure.4.4, the total number of ICs is 200 and NC_Rate is set to 0.8. Thus 160 ICs are classified as NCs. Then hosts A and B are added to FHL. As a result, ICs destined to host A and B are classified as NCs, and the other ICs are classified ACs.

LACD uses the FHL for outbound IC classification until the next IC logs analysis time. By analyzing IC logs in each CIT, LACD keeps on updating the FHL to adapt the changes of the communication patterns of the target host.

The ratio of LCs classified as ACs is about $1.0 - NC\_Rate$ if there is no worm in the

CWT. On the other hand, since Silent worm selects an infection target from the hit-list randomly, the ratio of WCs classified into ACs is $\frac{N_{all}-N_{friend}}{N_{all}}$ where N_{friend} is the size of FHL and N_{all} is the number of hosts in the network. Therefore, the ratio of WCs classified into ACs are several times larger than the ratio of LCs classified into ACs when NC_Rate is set to large enough, and N_{friend} is small enough compared to N_{all}.

4.2.4 AC Tree Detection

Here, how to detect AC trees in a distributed manner is described.

4.2.4.1 NC Notification Message Transmission

To detect AC trees, in addition to outbound ICs, each LACD needs to classify inbound ICs of its target host into either of ACs or NCs. Different from the outbound ICs, however, each LACD cannot classify inbound ICs directly. Therefore, the sender side LACD needs to help the classification at the receiver side. Here, *sender side LACD* is the LACD that monitors the source host of an IC.

When a host opens an outbound IC, the corresponding LACD sends a NC Notification Message (NNM) to the LACD that monitors the IC's destination host if the following conditions are both satisfied.

1. At the sender side LACD, the outbound IC is classified into a NC.
2. The sender side LACD has not sent a NNM to the receiver side LACD since the last IC classification.

A NNM indicates that an IC is classified into a NC by the sender side LACD. NNM includes the information about the corresponding IC and the expiration time. Usually, the expiration time is set to the next IC log classification time at the sender side LACD. After the expiration, the sender side LACD may no longer classify the IC into NC.

On receiving NNM messages, the receiver side LACD stores the messages as Inbound NC Caches. Each NC Cache is removed at its expiration time.

Then, when a LACD monitors an inbound IC, the IC is classified into a NC if the LACD has corresponding NC Cache; otherwise the IC is classified into an AC.

Here, an important issue for the sender side LACD is how to locate the IP address of the receiver side LACD. In this thesis, it is assumed that the sender side LACD sends NNMs to the destination hosts of outbound NCs and the receiver side LACD can capture and process the NNMs since the LACD monitors the destination hosts. The same approach is adapted to the transmission of other types messages.

4.2.4.2 Local AC Concatenation

To detect a part of an AC tree, LACD concatenates inbound/outbound ACs of its target hosts into groups, as shown in Figure.4.5. A group of concatenated ACs is named as a Local AC tree (LAT). Each LAT is a part of an AC tree and consisted of equals or more than one ACs. Figure.4.5 shows three LATs generated at host X.

There are two cases where a new LAT is generated. First, when LACD monitors a new inbound AC, the LACD always generates a new LAT with the AC. In Figure.4.5, LAT1 and 3 are generated when inbound ACs from Host A and F are monitored respectively.

Second, when LACD monitors a new outbound AC, a new LAT is generated if any inbound/outbound AC has not been monitored for a period of time. The period is named as *CLT (Connection Limit Time)*. CLT is a parameter to concatenate ACs. CLT is included in TH_Tuple as mentioned before. In Figure.4.5, CLT is set to T_{clt} and when an outbound AC is monitored at t_3, LAT2 is generated.

On the other hand, if there are LATs that contain ACs monitored within the past CLT, a new outbound AC is concatenated with the LATs. In Figure.4.5, outbound ACs monitored at t_1 and t_2 are concatenated with LAT1, and an AC at t_4 is concatenated with LAT2.

When a LAT has an inbound AC, the source host of the AC is named as the *parent host* of the LAT. In Figure.4.5, the host A is the parent host of LAT 1.

The longer the CLT is, the larger the size of each LATs is. When LACDs have multiple CLTs, LATs are generated for each CLT. Figure.4.6 shows an example. In this case, LACD has two CLTs: 2TU and 5TU and three LATs are generated in total: one LAT with CLT=5TU and two LATs with CLT=2TU. In this paper, all LACDs are assumed to use the same set of CLTs in order to concatenate LATs in different hosts to detect AC Trees.

Since the number of generated LATs keeps increasing over time, old LATs must be removed before they exceed the memory capacity of the LACD. In this paper it is assumed that LACDs have enough memory capacity and remove LATs in order of the generation time, and there is no detrimental effect by the removal. Also, since LATs can be generated only when ACs are transmitted, in most target hosts including some kinds of servers, the number of LATs LACDs must maintain at a time is considered not to be so high.

Here, since a LAT is a part of an AC Tree detected based on only the local information of each target host, LATs maintained by different LACDs will correspond to one AC tree. In Figure.4.7, 4 LATs correspond to an AC tree. Each LACD maintains the information of an AC tree, in which its target host is included, as a tuple named ACT_Info. Each ACT_Info consists of information about a LAT and the size of an AC tree, to which the LAT corresponds. The size of the corresponding AC tree is named as S_{ACT}. The initial

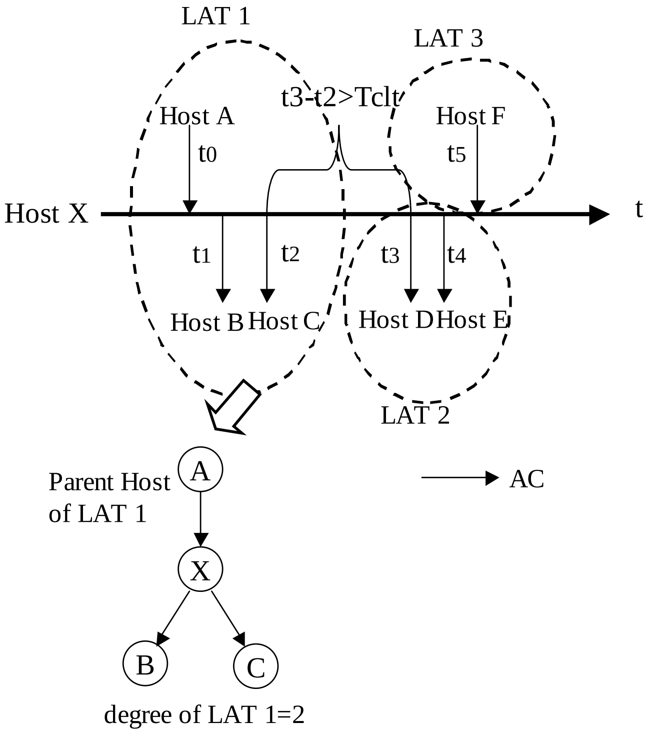


Figure 4.5: Local AC Concatenation

value of S_{ACT} is 1, and as new AC is added to a LAT, the corresponding S_{ACT} is increased by 1.

4.2.4.3 ACT Update Message Transmission

To estimate the actual size of detected AC trees in a distributed manner, LACDs that belong to an AC tree communicate to each others via AUMs(ACT Update Messages). An AUM indicates the update of an AC tree. Here, *update* means that an AC tree grows by adding a new host.

A LACD sends an AUM when S_{ACT} corresponding to a LAT is increased by a value named TH_{update}, and the LAT has an inbound AC. TH_{update} is a parameter to control the frequency of transmission of AUMs. For example, if TH_{update}=1, a new AUM is sent every time S_{ACT} is increased by 1 or more. If TH_{update}=2, a new AUM is not transmitted until S_{ACT} is increased by 2 or more. The destination of the AUM is the parent host of

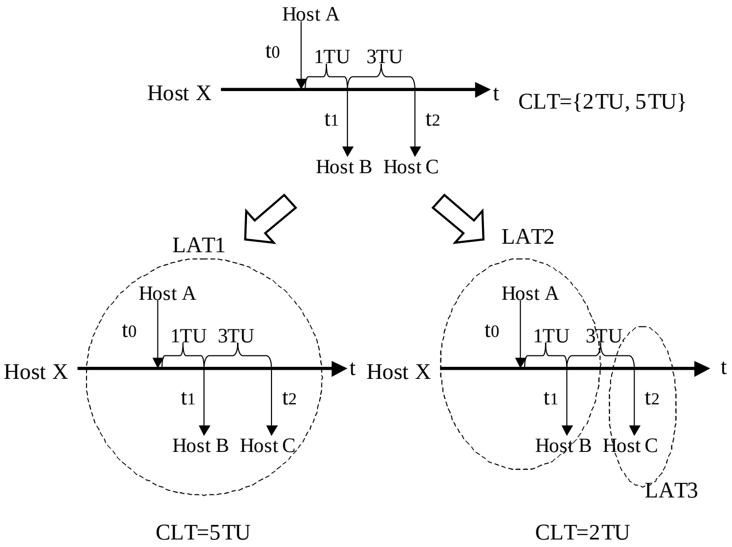


Figure 4.6: LATs with multiple CLTs

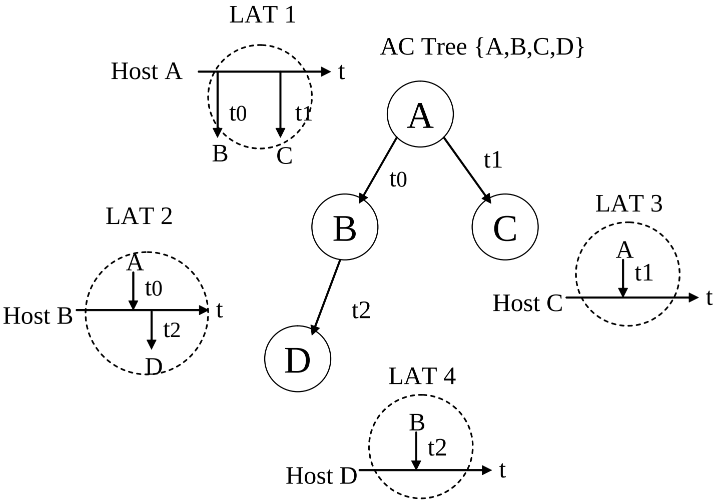


Figure 4.7: LATs corresponding to an AC tree

the LAT.

An ACT Update Message consists of three data: (1) the increased value of S_{ACT} since the last transmission, (2) CLT value (3) generation time of an AC that links the source and destination host (inbound AC of a LAT at the sender side). The AC in (3) links the sender and receiver side LACDs.

The receiver side LACD identifies a LAT that includes the AC, and adds the increased value in the AUM to S_{ACT} of the corresponding ACT_Info. Finally, if S_{ACT} exceeds the corresponding TH_{ac} in TH_Tuples, the LACD sends an alert to the network manager. In addition, the receiver sends a new AUM to its parent LACD recursively to propagate the update information further. Thus, AUMs are sent from children sides to parent's sides of an AC tree while the update information of an AC tree is aggregated at the LACD that monitors the root host of the AC tree.

Figure.4.8 shows an example. The AC tree is composed of four ACs from (1) to (4) and AUMs are sent to the parent side LACDs. In this figure, TH_{update} is set to 2, and with the ACs of (3) and (4), S_{ACT} of an ACT_Info at C_{LACD}(LACD of host C) is increased by 2. Then, as shown in (5), C_{LACD} sends an AUM with increased value=2 to B_{LACD}. On receiving the AUM, B_{LACD} adds 2 to S_{ACT} of a ACT_Info corresponding to the AUM.

As a result, S_{ACT} of an ACT_Info at B_{LACD} becomes 4. Then, an AUM with the increased value=3 is sent to A_{LACD}, which monitors the parent host of the LAT at host B. As a result, S_{ACT} at A_{LACD} becomes 5, and then the A_{LACD} can estimate the true size of the AC tree.

TH_{update} tunes the balance between the number of transmitted AUMs and detection speeds. If TH_{update}=1, LACDs can estimate the tree size accurately in real-time, but many AUMs need to be transmitted and the detection overhead will be high. On the other hand, if TH_{update} is set to too large, it will take a long time for the detection although the number of transmitted AUMs can be small.

Here, a LAT that does not have a parent host is named *root LAT*. In Figure.4.8, the root LAT is at host A. Since AUMs are transmitted from children to parents LACDs, S_{ACT} of the root LAT will be the largest value among the all S_{ACT}s corresponding to the AC tree. Therefore, the S_{ACT} corresponding to the root LAT will be closest to the actual size of the AC tree.

4.2.5 Detection and Threshold Adjustment

Each time a new AC is detected or a new AUM is received and processed, LACD checks whether the corresponding S_{ACT} exceeds a corresponding TH_{ac} in TH_Tuple. If S_{ACT} exceeds the TH_{ac}, LACD estimates the existence of worms and sends an alert to the network manager. On receiving the alert, the network manager investigates the network

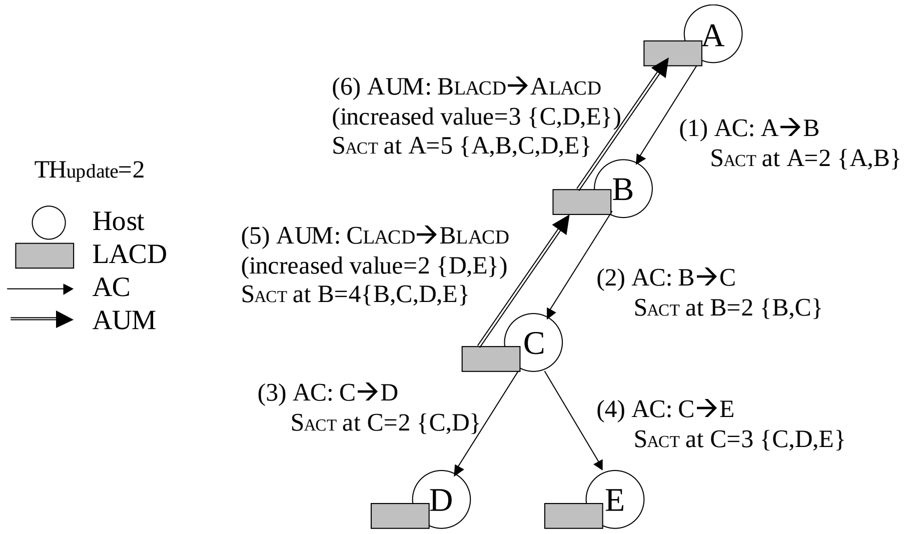


Figure 4.8: ACT Update Message

and hosts. After T_{invest} passes, it becomes clear whether the alert is a true or false alert. If the alert is found to be a false alert, the network manager sends a false alert notification message (FANM) to the LACD.

Here, various information will be required for the investigation depending on the situations. Since the focus of d-ACTM (d-ACTM/VT) is not the verification of alerts but the detection of worms, however, the network manager is regarded as just an oracle that has an ability to verify alerts, and don't discuss about the details of the knowledge used by the network manager.

In d-ACTM, T_{invest} represents a delay from when a LACD raises a false alert to when the LACD increases the corresponding TH_{ac}. Although it is difficult to estimate the accurate time needed for investigation, T_{invest} will be much longer than the worms' infection intervals.

Next, how to determine TH_{ac}s is shown. LACD has a parameter, named desirable false positive alert interval (DAI). DAI indicates the acceptable frequency of false positive alerts issued by each LACD. For example if DAI=$1.0 * 10^6$TU, the average interval of two continuous false alerts raised by a LACD must be no less than $1.0 * 10^6$TU. The purpose of DAI is to maintain the desirable balance between false alert rate and detection speeds. The desirable false alert interval per TH_Tuple, named DAI_{TUP}, is computed as $DAI * NUM_{TH\_Tuple}$. For example when DAI=$1.0 * 10^6$ and the number of TH_Tuples is 3 (as Figure.4.3 shows), DAI_{TUP} is $3.0 * 10^6$TU. Also, when there are N LACDs (=N

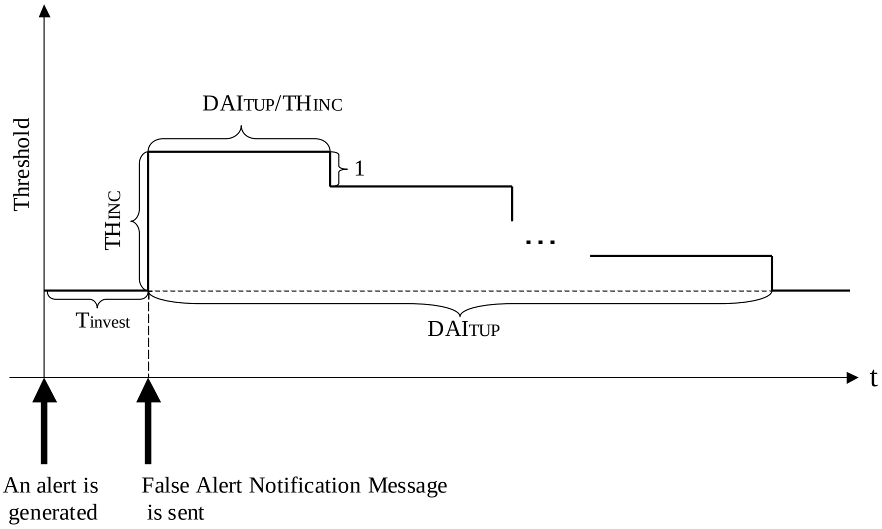


Figure 4.9: Threshold Adjustment

hosts) in a network, in each $DAI * N$ TU, one alert will be raised in the network on average.

To bring interval of its continuous false alerts close to DAI, each LACD autonomously adjusts its TH_{ac}s. In summary, LACD increases a corresponding TH_{ac} when it turns out that its alert is found to be a false alert, and decreases the TH_{ac} when no false alert has been generated for a certain period of time.

When a LACD starts operation, each TH_{ac} is set to an initial value. Then, the LACD keeps adjusting the TH_{ac}s during its operation. As Figure.4.9 shows, when a FANM is received from the network manager, the TH_{ac} that causes the false alert is increased by TH_{INC}. On the other hand, when no false alert has been generated for more than DAI_{TUP}/TH_{INC} according to a CLT, LACD judges that the corresponding TH_{ac} is too high and decreases it by 1.

4.3 d-ACTM/VT

In this section, d-ACTM/VT (d-ACTM with distributed Virtual AC Tree Detection) that extends the current d-ACTM by introducing the distributed VAC tree detection is proposed.

4.3.1 Tree Aggregation Approach

Figure.4.10 shows the approach of distributed AC tree aggregation. In Figure.4.10, ACT2 and ACT3 are aggregated with ACT1 as a VAC tree.

When the size of an AC tree becomes equal or greater than a threshold named TH_{TUM}, the LACD that monitors the root host of the tree sends TUM (Tree Upload Message) to the LACD that monitors the host which has opened NCs to senders target host within a certain period of time.

TUM includes (1) the size of an AC tree rooted at the target host, (2) NC that links the sender/receiver LACDs, and (3) *TTL*. (1) is denoted as S_{TUM}. TTL is a parameter of LACD that specifies the range of destinations of TUM. Initially, TTL is set to a value named TTL_{INI}.

As a TUM is received by a LACD, its TTL is decreased by 1. Then, if the TTL is still greater than 0, the receiver recursively transmits the TUM to upper side LACDs. In Figure.4.10, TTL_{INI}=2 and a TUM is transmitted from LACDs of host F to B through host D.

When a LACD receives a TUM and its target host is included in an AC tree as a leaf or internal node, the LACD sends a new message named TRM (Tree Relay Message) to the LACD of the parent host in the AC tree. TRM includes (1) a received TUM and (2) information about an AC that links the sender and receiver LACDs. The TRM is relayed to the root of the tree recursively. In Figure.4.10, a TRM is relayed from LACDs of host E to host A through host C.

When a LACD receives a TUM or TRM and its target host is a root of an AC tree T, it stores the TUM or TRM. Then, the sum of the size of T and S_{TUM}s in the TUMs and TRMs is used as the size of a VAC tree. If the size exceeds a threshold named TH_{vac}, the LACD detects the existence of worms and sends an alert to the network manager. In Figure.4.10, at the LACD of host A, the sum of S_{TUM}s in TRMs from host B and C are 7(=4+3), and the size of ACT1 is 4. Thus, the size of the VAC tree consisted of ACT1-3 is 11.

In the following sections, more detailed algorithms in LAT level will be given.

4.3.2 Division in LAT level

As shown in Figure.4.7, in d-ACTM/VT, an AC tree is composed of some LATs, which are managed by different LACDs. In other words, each LAT corresponds to an AC tree. Thus, in LAT level, when tree division occurs, LATs that should correspond to one tree are forced to correspond to different small trees. There are two types of LAT division; (1) Remote LAT division and (2) Local LAT division as Figure.4.11 shows.

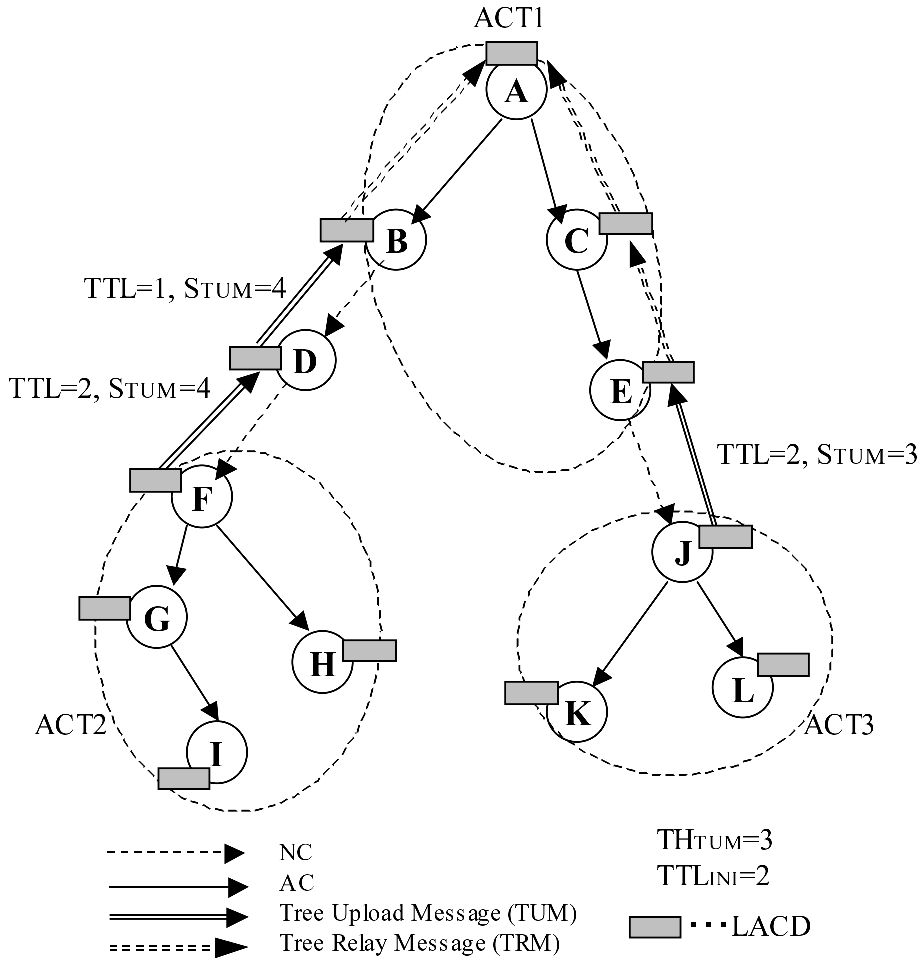


Figure 4.10: Approach of Distributed AC Tree Aggregation

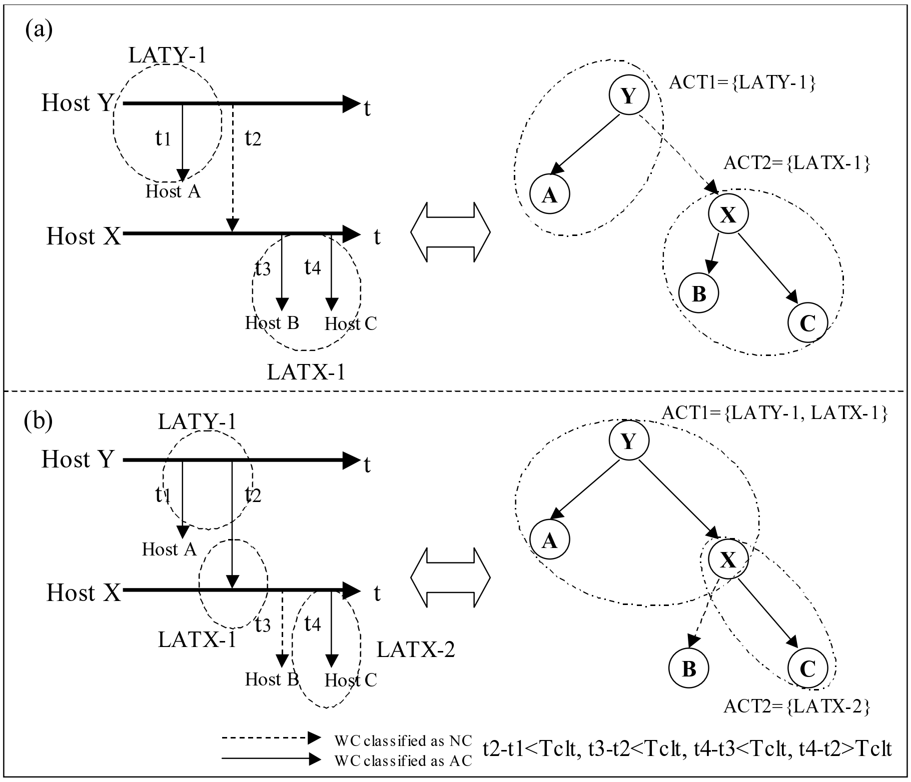


Figure 4.11: (a)Remote LAT division, (b)Local LAT division

In remote LAT division, LATs at different hosts correspond to different AC trees. In Figure.4.11(a), due to the tree division by a WC at t2, LATY-1 and LATX-1 correspond to different trees ACT1 and ACT2

In local LAT division, LATs at the same host correspond to different AC trees due to outbound WCs classified as NCs. In Figure.4.11(b), since a WC at t3 is classified as a NC and $t4 - t2 > T_{clt}$, LATX-1 and LATX-2 correspond to different trees, ACT1 and ACT2 respectively.

Thus, there are two types of LAT aggregation; Remote LAT Aggregation and Local LAT Aggregation.

4.3.3 Remote LAT Aggregation

4.3.3.1 Sender side of TUM

Figure.4.12 shows the transmission of TUMs for the aggregation of LATs in different hosts.

LACD sends TUMs when S_{ACT} corresponding to a LAT becomes equals or greater than TH_{TUM}, and the LAT is a *Root LAT*. Here, Root LAT is a LAT that only contains outbound ACs. In Figure.4.12, with ACs at t4, t5 and t6, S_{ACT} of LATX-1 becomes 3, which is equals to TH_{TUM}. Thus, TUMs are to be transmitted.

Here, some notations are introduced. R denotes a root LAT, S_{ACT} of which is equals or greater than TH_{TUM}. In Figure.4.12, LATX-1 is R. Also, C_L^{first} and C_L^{last} denote the first and last AC of a LAT L respectively. C^T denotes an IC generated at time T. Also, a function $G(C)$ returns the generation time of an IC C.

The destination of a TUM is specified based on an IC named C^{base}. At first, C^{base} is set to the C_R^{first}. In Figure.4.12, C^{base} is C^{t4}.

A LACD sends a TUM to a LACD monitoring a host that has sent a NC named C^{in} to its target host if C^{in} satisfies the condition below.

- C^{in} is linked with C^{base} by a sequence of ICs named IC_Chain, and the size of the IC_Chain is smaller than TTL.

The TTL of a TUM is decreased by the size of the IC_Chain before transmission.

Here, an IC_Chain is a sequence of ICs sent or received by a same host. IC_Chain consists of two arbitrary ICs as the first and last elements and the minimal number of outbound NCs generated between the ICs. In the IC_Chain, intervals between any adjacent elements need to be no longer than CLT. The number of outbound NCs within the IC_Chain is defined as the size of an IC_Chain. When two arbitrary ICs belong to a same IC_Chain as the first and last elements, the ICs are *linked* by the chain.

In Figure.4.12, the sender host of C^{base} is Host X. Then, C^{t0}, C^{t1},C^{t3} can be C^{in}. In the case of C^{t1}, C^{t1} is directly linked with C^{t4} since $t4 - t1 \leq T_{clt}$, and therefore C^{t1} is a C^{in}. Then, a TUM with TTL=2 is sent to the LACD that monitors the sender of C^{t1}, Host Y. The same analysis holds for C^{t3}.

In the case of C^{t0}, C^{t0} is linked with C^{t4} by an outbound NC C^{t2} since $t4 - t2 \leq T_{clt}\ and\ t2 - t0 \leq T_{clt}$, and therefore satisfies the condition above. Then, a TUM with TTL=1 is sent to the LACD of host V, the sender of C^{t0}.

4.3.3.2 Receiver side of TUM

On receiving a TUM, the receiver LACD decreases TTL of the TUM by 1. Then, the LACD associates a NC C^{in} in the TUM with a LAT L that satisfies one of the following conditions.

1. $G(C_L^{first}) \leq G(C^{in}) \leq G(C_L^{last})$
2. L is a root LAT and there is a chain that links C^{in} and either of C_L^{first} or C_L^{last}.
3. L is not a root LAT and there is a chain that links C_L^{last} and C^{in} $s.t.$ $G(C_L^{last} \leq G(C^{in})$.

In either case of (2) or (3), the chain size needs to be no greater than TTL of the TUM.

In Figure.4.12, host Y is a sender of C^{t1}, which is one of C^{in}s. Then, LATY-1 satisfies (3) since $t1 - C_{LATY-1}^{last} \leq T_{clt}$. Thus, since LATY-1 has an inbound AC C^{t7}, a TRM that includes the TUM is transmitted to host B, a parent host of host Y.

On the other hand, LATY-2 satisfies (2) since $C_{LATY-2}^{first} - t1 \leq T_{clt}$. The TUM is stored in the ACT_Info corresponding to LATY-2.

At a root LAT, the size of the VAC tree is computed. In Figure.4.12, the size at LATY-2 is 5(=2+3). Then, if the size exceeds TH_{vac}, an alert is raised.

Thus, in this example, the aggregation of LATX-1 and LATY-1, and aggregation of LATX-1 and LATY-2 are generated.

The LACD at host A sends TUMs each time S_{ACT} of LATX-1 is increased. Thus, when a LACD receives a new TUM corresponding to LATX-1, the LACD removes old TUM corresponding to LATX-1.

Finally, if the TTL is still greater than 0, the TUM is transmitted recursively. In this case, C^{base} is set to C^{in}. In Figure.4.12, since $t1 - t10 \leq T_{clt}$, the TUM is transmitted from host Y to host W recursively.

4.3.4 Local LAT Aggregation

As well as remote LAT aggregation, local LAT aggregation is also generated when S_{ACT} of a root LAT exceeds TH_{TUM}. Figure.4.13 shows an example. In the local aggregation, root LAT R is aggregated with LAT L generated before R if there exists a *IC_Chain* that links C_R^{last} and C_L^{first}, and the size of the chain is not greater than TTL_{INI}.

In Figure.4.13, LATX-1 and LATX-2 are linked by an outbound NC C^{t3}, and LATX-1 and LATX-3 are linked by C^{t1} and C^{t3}. Thus, the aggregation of LATX-1 and LATX-2, and the aggregation of LATX-1 and LATX-3 are generated. At LATX-2 and LATX-3, S_{ACT} of LATX-1 is treated like S_{TUM} and used for the computation of VAC trees.

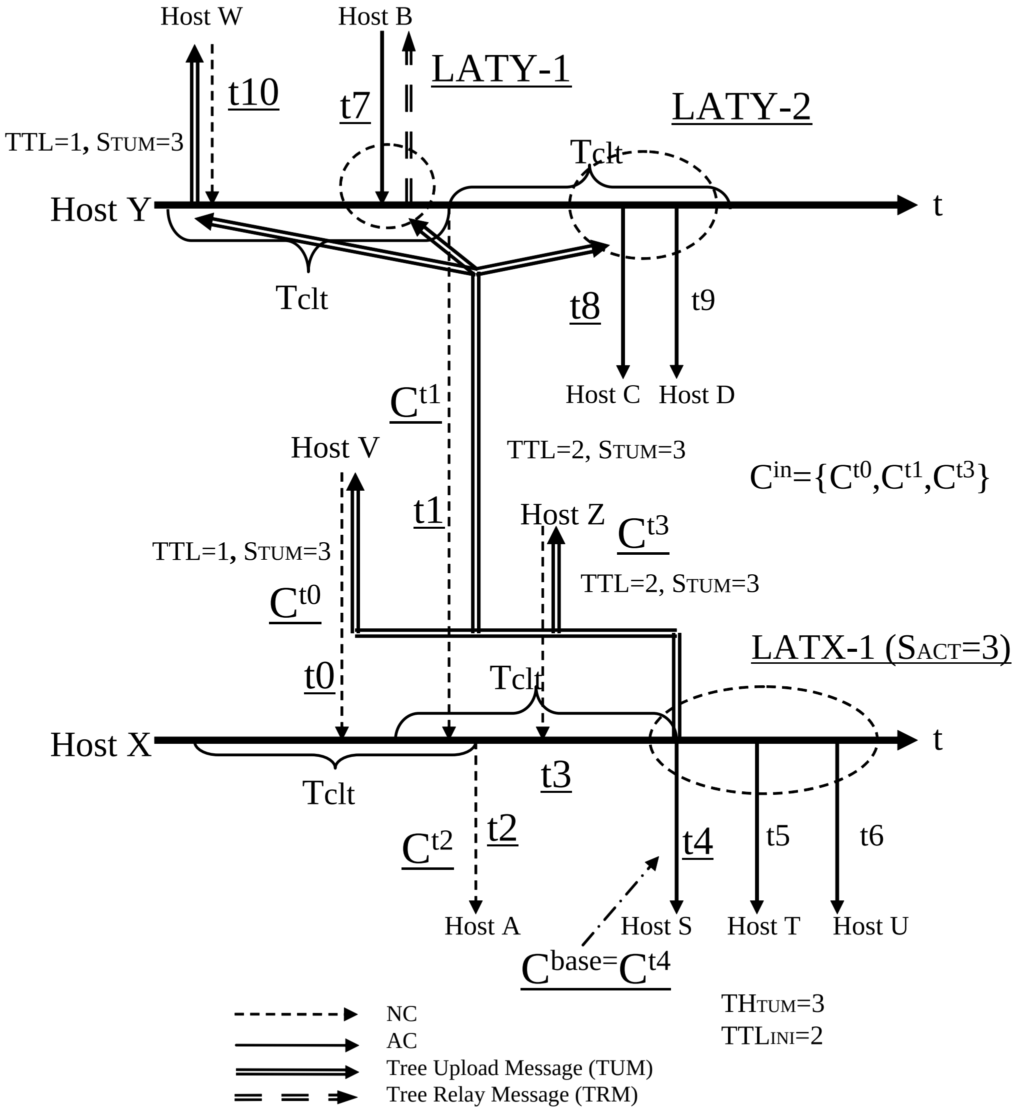


Figure 4.12: Remote LAT Aggregation

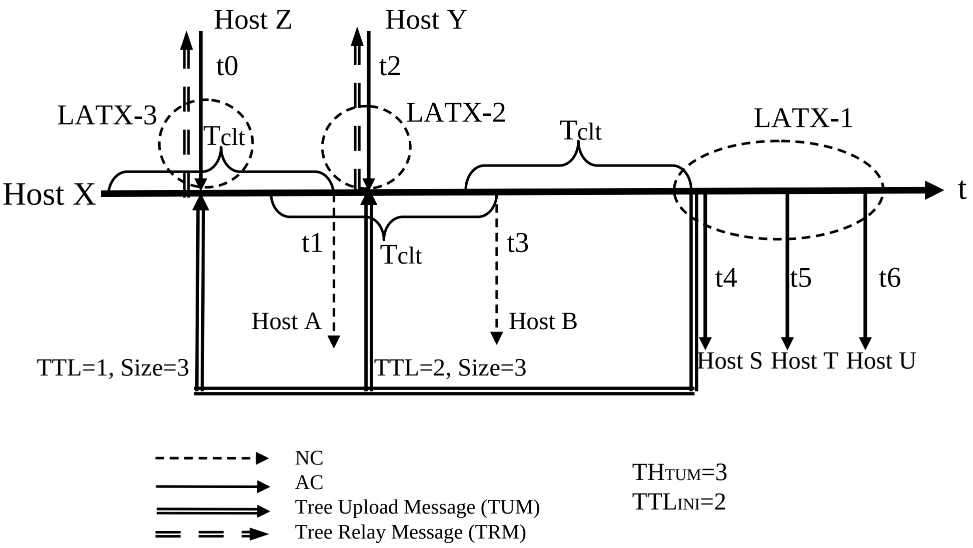


Figure 4.13: Local LAT Aggregation

Next, if L has an inbound AC, the LACD sends a message that includes S_{ACT} of R to the source of the AC. The message is named as TRM-local. In Figure.4.13, two TRM-locals are transmitted from LATX-2 and LATX-3 to host Y and Z respectively. Then, as similar to TRM, TRM-local is relayed to a root of an AC tree and used for the VAC tree detection.

4.3.5 Time Range of NC Logs

LACD needs to preserve logs of NCs in a range for tree aggregation. When T_R^{first} denotes the generation time of the oldest root LAT stored in the LACD, T_L^{first} denotes the generation time of the oldest non-root LAT, T_R^{latest} denotes the generation time of the latest root LAT, and T_C^{latest} denotes the generation time of latest appended outbound AC, the range is roughly expressed as

$$\begin{array}{lll}[& Min(T_R^{first} - CLT * (TTL_{INI} - 1), T_L^{first}) & , \\ & T_C^{latest} + CLT * (TTL_{INI} - 1) &]\end{array}$$

for outbound NCs, and

$$[T_R^{first} - CLT * TTL_{INI}, T_R^{latest}]$$

for inbound NCs.

If a LACD sends/receives many NCs, the size of logs in the range may exceed the storage capacity. In this case, some logs need to be removed. In this paper, it is assumed that LACDs have enough memory capacity, and removal of logs in the range does not occur. While how to remove the logs without any detrimental effect is one of the future works, FIFO strategy, in which older logs are removed at first, will be good.

4.3.6 Threshold Determination

Here the determination of two thresholds TH_{vac} and TH_{TUM} is described.

First, the determination of TH_{vac} is same as that of TH_{ac}. Since there are two detection thresholds (TH_{ac} and TH_{vac}), when DAI=T_{DAI}, both the interval of false alerts due to the VAC tree detection and AC tree detection should be $2 * T_{DAI}$. Then, both thresholds are to be adjusted to satisfy $2 * T_{DAI}$.

Second, TH_{TUM} is determined as

$$TH_{TUM} = TH_{ac} \times R_{TUM} \tag{4.1}$$

where $0 \leq R_{TUM} \leq 1$. The idea behind the expression is that the size of divided sub tree will be increased to a certain ratio of TH_{ac} if the tree is caused by worms propagation. As TH_{TUM} increases, the number of transmitted TUMs and TRMs are decreased.

Thus, in the VAC tree detection, two parameters R_{TUM} and TTL_{INI} have the major effect on the detection performance.

4.4 Evaluation

In this section, the computer simulation to evaluate the performance of d-ACTM/VT is described.

4.4.1 Simulation Model

In this simulation, an enterprise network where all internal hosts have vulnerabilities exploited by Silent worms is assumed. Figure.4.1 shows the default parameters of the simulation. Here, TU denotes a time unit.

After the detection thresholds become to satisfy a desirable false alert interval specified by DAI, a Silent worm infects one randomly selected host and starts propagation.

The details of the model are given in the following sections.

4.4.1.1 Network Model

In an assumed enterprise network, common client-server type network services such as SSH, Windows RPC Services, Web, Mail are in operation, but P2P applications, which cause tree like connection structures, are not run. Actually, many organizations prohibit the use of such P2P applications due to security reasons. The number of hosts is 500.

Patterns of the destination hosts of LCs in the network are modeled from 2 viewpoints. Model (1) focuses on how biased the destination hosts of LCs opened by each individual host are. Model (2) focuses on the bias of the destination hosts of LCs opened by all hosts in the network.

As for model (1), the destination hosts of each individual host are classified into 2 groups: Frequent Communication Hosts which include x % of all hosts in the network, and Infrequent Communication Hosts that include the other hosts. Then, 80% of LCs of each host are destined to its Frequent Communication Hosts and the others are for Infrequent Communication Hosts. Each host evenly opens LCs for the hosts in its Frequent Communication Hosts. The same holds for the hosts in its Infrequent Communication Hosts. As a default, x=24%.

As for model (2), the members of Frequent Communication Hosts of a host are randomly selected from all hosts so that the each host receives LCs from the other hosts with a same frequency from each others. Although the model may be somewhat different from typical networks with client-server services, this model is more difficult setting for worm detection sides since a kind of IDSes that focus on the occurrence of many connections destined from server to client hosts in the event of worms propagation [28] cannot be applied. Note that, however, the detection performance of d-ACTM/VT is almost same in networks where hosts are either of the clients or server type hosts.

As for the open interval of outbound LCs, in order to evaluate the characteristic of d-ACTM/VT under a basic network model, it is assumed that the open frequencies of all hosts are almost same. Then, the interval between two continuous LCs opened by all hosts follows the exponential distribution and the average is 10TU.

4.4.1.2 Silent Worm Model

As a worm model, a Silent worm that has address lists of all vulnerable hosts in the network is assumed. The worm infects hosts by exploiting the vulnerabilities of the services run on the hosts. The number of infection trials per each infected host is limited to 2 at most to evade detection methods that focus on the connection rate [9]. The infection interval of the worm is set to 10TU, which is equal to the average interval of outbound LCs. Here, the infection interval is the average interval between the generation times of 2 continuous

WCs opened by an infected host. The interval between when a host is infected and when the host starts infection activity is also 10 TU.

4.4.1.3 LACD Setting

NC_Rate of all LACDs are set to 0.8, with which the detection performance of d-ACTM/VT against the destination bias model explained in 4.1.1 is optimized. Here, in reality, each host may have the different bias of destination hosts from each other, and the bias can be changed as time advances. Thus, it is desirable that each LACD can automatically adjust the NC_Rate to an optimized value based on the activity of its target host, and this is one of the future works.

Next, as a default setting, every LACD uses one TH_Tuple with CLT=10TU. Therefore, d-ACTM/VT can detect worms with infection interval shorter or equal to the average interval of outbound LCs. Since DAI is set to $5 * 10^6$TU, in every 10^4 TU, 1 false alert is generated in the network. If 1TU=1sec, 1 false alert will be generated in every about 3 hours, and which is considered to be a reasonable rate.

In the following sections, the simulation results are shown.

4.4.2 The number of infected hosts

Figure.4.14 shows the number of infected hosts before detection with d-ACTM/VT and d-ACTM as a function of DAI.

With DAI=$5 * 10^6 TU$, d-ACTM/VT detects worms when 36 hosts are infected, and reduces the infected hosts by more than 20% compared to d-ACTM. This result indicates the effectiveness of distributed VAC tree detection.

4.4.3 Relation between the number of transmitted messages

Figure.4.15 shows the number of infected hosts and the average number of messages transmitted by each LACD of d-ACTM/VT as a function of R_{TUM}. The message contains TUMs, TRMs, NNMs and AUMs. As R_{TUM} increases, the number of transmitted TUMs and TRMs is decreased, and as a result, the number of infected hosts is increased. With R_{TUM} =0.35, the number of transmitted messages is $3.2 * 10^5 TU$. Since the average number of outbound ICs that each host opens in the simulation period is about $4 * 10^6$, the number of transmitted messages among LACDs is about 8% of the that of outbound ICs.

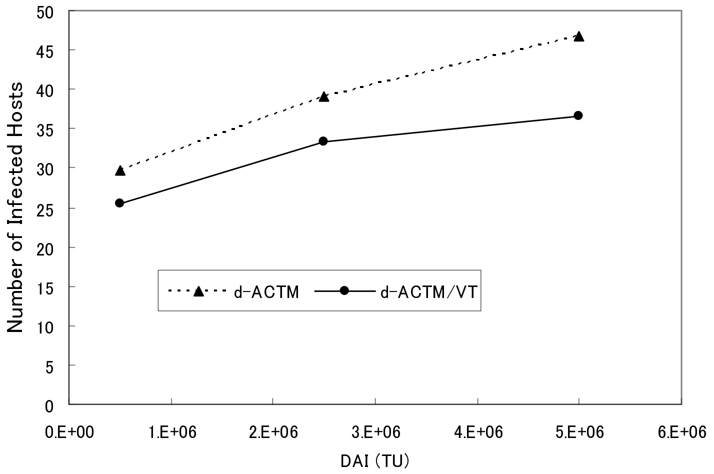


Figure 4.14: The number of infected hosts before d-ACTM/VT detects worms

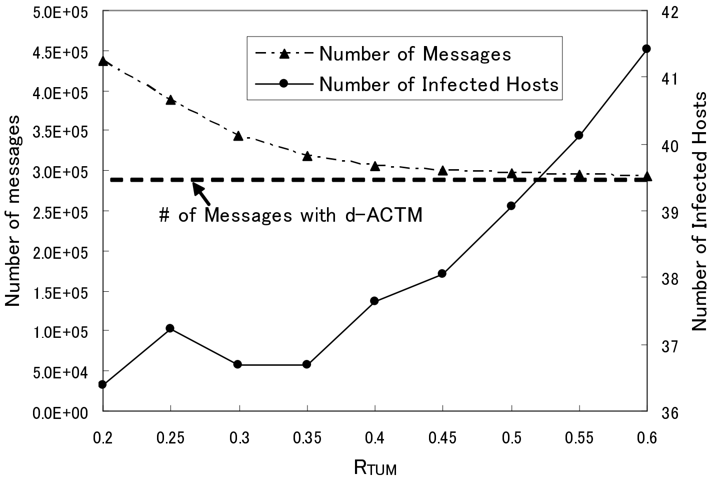


Figure 4.15: Relation between the number of transmitted messages and infected hosts

Table 4.1: Default Simulation Parameters

# of hosts	500
Simulation Time (TU)	$4 * 10^7$
Interval of outbound LCs (TU)	10
Ratio of hosts in Frequent Communication Hosts	0.24
Ratio of # of LCs to Frequent Communication Hosts	0.8
# of infection trials per instance	2
Infection interval (TU)	10
NC_Rate	0.8
TH_{update}	1
CLT(TU)	10
Initial TH_{ac}	10
Initinal TH_{vac}	10
TTL_{INI}	1
R_{TUM}	0.35
DAI(TU)	$5 * 10^6$
TH_{INC}	5
CIT (TU)	$5 * 10^4$
CWT (TU)	$5 * 10^4$
T_{invest} (TU)	10^4

4.4.4 The effect of TTL_{INI}

Figure.4.16 shows the number of infected hosts and the average number of messages as a function of TTL_{INI}. As TTL_{INI} increases, the number of transmitted messages is increased. As to the number of infected hosts before detection, with $TTL_{INI} = 2$, the number is slightly reduced. With TTL_{INI}=3, the number is steeply increased by 2. Thus, $TTL_{INI} = 1$ achieves both the fast detection and small network overhead.

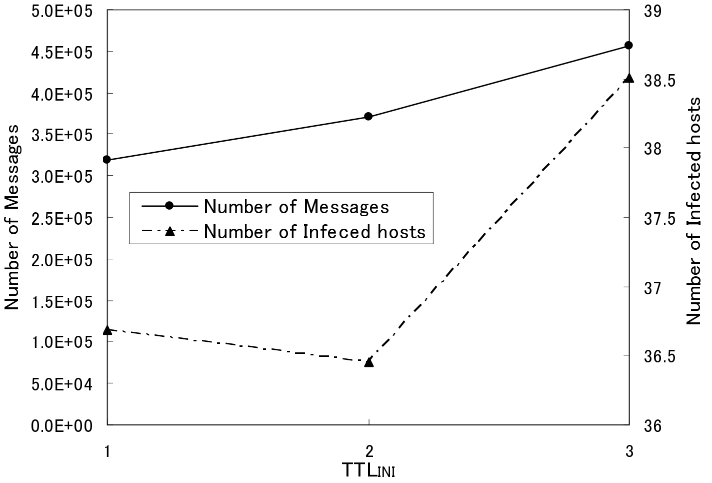


Figure 4.16: The effect of TTL_{INI} on the detection performance

4.4.5 The effect of the bias of the destination hosts of outbound LCs

Figure.4.17 shows the number of infected hosts as a function of the ratio of the number of hosts in Frequent Communication Hosts. As the figure shows, when the ratio of hosts in Frequent Communication Hosts is smaller than 0.3, d-ACTM/VT can detect worms before 10% of hosts are infected. Here, in many networks, the most hosts frequently connect to only a few percent of hosts in the network [25] [26]. Therefore, from the viewpoint of the bias of the destination hosts of LCs, d-ACTM/VT is effective in most networks.

4.4.6 The effect of worm's infection intervals on the detection performance

Figure.4.18 shows the number of infected hosts as a function of worm's infection interval. Figure.4.18 also shows a case where LACDs use two TH_Tuples with CLT=6 and 10 TU. From the figure, d-ACTM/VT can detect worms with infection interval $\leq$ 10TU before 10% of all hosts are infected.

Here, the intervals of most existing worms is much shorter compared to the average interval of LCs [9]. Thus, from Figure.4.17 and Figure.4.18, d-ACTM/VT is effective against the most of Silent worms in the most networks. In more details, d-ACTM/VT

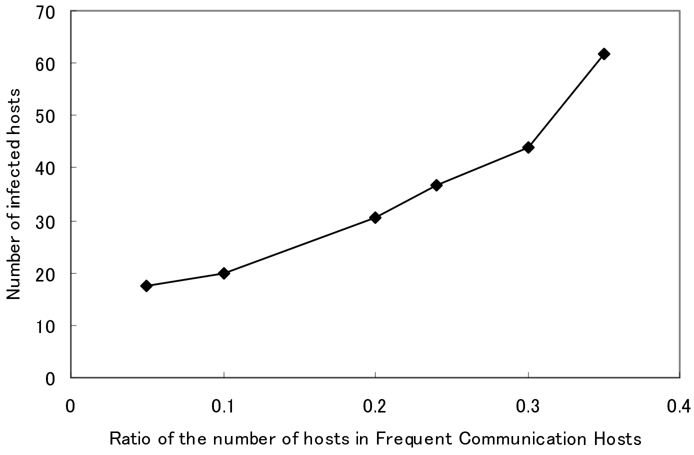


Figure 4.17: The effect of the ratio of hosts in Frequent Communication Hosts

can detect worms with infection interval ≤ 10TU before 10% of hosts are infected in the network where 80% of outbound LCs of each host are destined to 24% of all hosts. Considering it is difficult for existing methods to detect Silent Worms effectively, the detection performance of d-ACTM/VT is quite promising.

With CLT=10TU, as the infection interval decreases, the infected hosts are increased, except the point where the infection interval decreases from 6TU to 5TU. The reason is that when a WC is classified as NC, its previous and next WCs can be concatenated if the infection interval is equal or less than 5TU.

With CLT=6 and 10TU, the number of infected hosts with the worm's infection interval≤ 6TU is smaller compared to the case where only CLT=10TU is used. Since the detection thresholds corresponding to CLT=6TU is smaller than the thresholds corresponding to CLT=10TU, worms with infection interval ≤ 6TU can be detected faster using CLT=6TU compared to the case with CLT=10TU.

4.4.7 The effect of the number of infection trials per instance

Figure. 4.19 shows the number of infected hosts as a function of the number of infection trials per instance. As the figure shows, the number of trials of each infected host does not influence the performance of d-ACTM/VT.

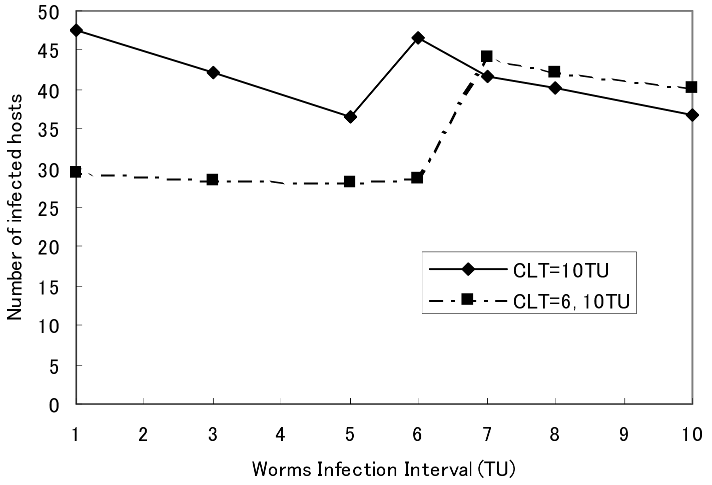


Figure 4.18: The effect of worms infection interval

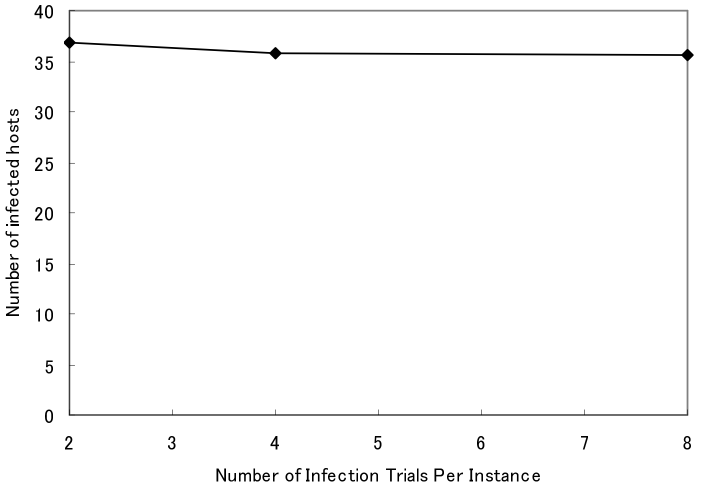


Figure 4.19: The effect of number of infection trials Per Instance

4.4.8 Comparison with ACTM

ACTM, which requires a central server for detection, detects the existence of worms before 37 hosts are infected with a condition similar to Table 4.1. Thus, d-ACTM/VT can detect worms as fast as ACTM without any global knowledge of the network.

As to the network overhead, the number of IC logs transmitted to the ACTM server from several packets capture devices in the network is 50 per 1TU. With d-ACTM/VT, on the other hand, the number of messages transmitted among LACDs is 4 per 1TU. This is because, in d-ACTM/VT, original IC logs are processed at each LACD and only the summarized data are transmitted in the network.

Next, as to the computation cost for detection, the computation cost of the ACTM server is proportional to the number of hosts in the network. With d-ACTM/VT, on the other hand, each LACD only need to analyze a part of AC trees in which its target host is included. This means, the computation cost of each LACD is scalable from the viewpoint of the number of hosts in the network. In addition, the ACTM server regards each AC tree as a group of sub trees like LATs, and therefore the analysis approach is similar to d-ACTM/VT. Thus, the computation cost of the ACTM server and the total costs of all LACDs are not so different except that, with d-ACTM/VT, the exchange of information of LATs among LACDs causes some network transmission costs. Therefore, the computation cost of each LACD is smaller compared to the ACTM server.

The same discussion holds for the data storage size required for the detection. The storage size of each LACD does not depend on the number of hosts in the network.

Thus, from the viewpoint of network overhead, computation cost and storage size, d-ACTM/VT is more scalable compared to ACTM.

4.4.9 Comparison with other approaches

Figure. 4.20 shows the performance comparison with other graph based approaches. As this figure shows, d-ACTM/VT can detect worms more than 10 times faster than GriDS and 4 times faster than T.K.

4.4.10 Threshold Values

Figure. 4.21 shows the average TH_{vac}, TH_{ac} of all hosts as a function of time. At $9.0 * 10^6$ TU, thresholds become stable and the average interval between continuous false alerts become 10^4 TU. The stable values of TH_{vac} and TH_{ac} are 20.5 and 26.5 respectively. By comparison, in the case of d-ACTM where only TH_{ac} is considered, at $6.5 * 10^6$ TU, TH_{ac} becomes stable. The reason is stated in below.

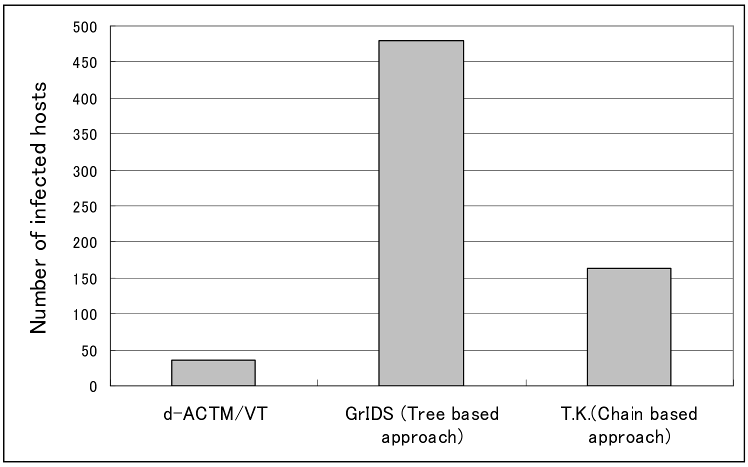


Figure 4.20: Comparison with other approaches

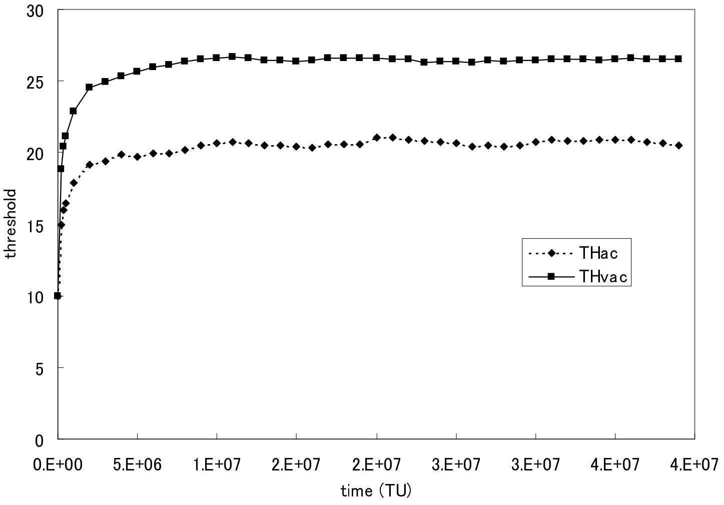


Figure 4.21: Detection Thresholds

4.4.11 Cost of VAC tree detection

The introduction of VAC tree detection involves some costs such as (1) the increase of the number of transmitted messages, (2) the increase of stored inbound/outbound NC logs, (3) the extension of time to adjust detection thresholds to optimal values.

As for (1), the number of increased messages is small as mentioned in 4.2.3, and therefore the increase of the network overhead will not be a serious problem.

As for (2), since inbound/outbound NC logs are used for VAC tree detection, the storage size needed for detection is increased. In this simulation, the number of logs stored at a time is about 20. Since the size of each NC log is small, the required storage size for NC logs is small enough.

As for (3), since d-ACTM/VT uses two detection thresholds, it takes longer time for each LACD to adjust its detection thresholds to optimal values compared to d-ACTM. Here, the optimal value is a value of threshold, with which the interval of continuous false alerts becomes almost equal to DAI. If thresholds are smaller or larger than optimal values, more false alerts will be generated or the detection speed will be slower than d-ACTM/VT expects, respectively. As a whole, the extension of time taken for the adjustment is undesirable. In the simulation, with d-ACTM/VT, it takes 1.4 times longer time until the average interval of false alerts become 10^4 TU compared to d-ACTM. It is considered that, however, the time can be shorten through the improvement of the adjustment algorithms such as the exchange of thresholds values among LACDs. This will be one of the future works.

Therefore, the cost of introduction of VAC tree detection is not serious or able to be addressed.

4.4.12 Discussion about hosts with high frequent communication

In this simulation, LC open interval of all hosts is assumed to take the same value. Here, in the case where some hosts open LCs with much higher frequency compared to the other hosts, detection thresholds of d-ACTM/VT are increased and then, more hosts will be infected before detection. One solution to address the problem is to exclude hosts with high frequent communication from the detection targets. How to identify and exclude such hosts in a distributed manner is one of the future works.

4.4.13 The border between ACTM and d-ACTM/VT

Here, a discussion about the cases where a network should introduce d-ACTM/VT instead of ACTM is given.

As discussed in 4.4.8, the detection performance of ACTM and d-ACTM/VT are almost equal, and d-ACTM/VT is more scalable than ACTM in terms of the computation and network overheads. In addition, since large networks will not have any vantage points where all internal traffic can be observed, in the case of ACTM, many traffic capture devices are needed to be deployed in various points to gather all IC logs. Then, if capture devices are deployed in a way where every host has any devices that monitor its all inbound and outbound ICs, devices can be used as LACDs by adding IC classification and AC/VAC tree detection modules. If ICs can be directly processed at their captured points instead of central IDSes, the network traffic due to the worm detection is significantly reduced.

Moreover, since LACDs can be incrementally deployed, d-ACTM/VT is suitable for the network where there is no absolute security authority which controls the whole network security, and each segment in the network wants to enters/leaves security cooperation in the network at its will.

Thus, d-ACTM/VT can be preferable to ACTM for the network where any of the following conditions are satisfied.

1. There are too many ICs to be analyzed by a few central servers at a place. Also, the costs to deploy capture devices in a way where every host has any devices that monitor its all traffic, and to modify the capture devices to LACDs are not high.
2. There is no absolute security authority in the network, and each segment wants to join security cooperation in the network at its will.

On the other hand, if neither of the conditions is satisfied, ACTM is considered to be more preferable.

4.5 Summary

To detect Silent worms in a distributed manner, this chapter has proposed Distributed Anomaly Connection Tree Method (d-ACTM) and d-ACTM/VT (d-ACTM with Virtual AC tree detection). By detecting AC and VAC trees in a distributed manner, d-ACTM/VT realizes the fast detection of such worms.

Through the computer simulation experiments, it has been shown that d-ACTM/VT can detect Silent worms before 10% of all vulnerable hosts are infected under the condition where the infection interval is equal or less than the normal connection interval and there are 500 vulnerable hosts in the network. With d-ACTM/VT, the number of messages transmitted in the network is 8% compared to ACTM where a central server needs to process all logs from various packets capture devices. Thus, d-ACTM/VT is more scalable in terms of network overhead, computation cost and storage requirements compared to ACTM. Also, it is made clear that d-ACTM/VT can reduces the number of infected hosts by 20 % with a small increase of network overhead compared to d-ACTM.

Chapter 5

Conclusion

Recently, the appearances of hit-list worms have been widely discussed among network security community. Nevertheless, existing methods are not enough to effectively detect the worms. The theme of this thesis is how network based IDS can detect the existence of the hit-list worms.

Chapter 1 describes the background of computer worms and explains the contribution and focus of this research.

Chapter 2 introduces the taxonomy of computer worms and existing detection methods in detail. Then, the problems with existing methods and the position of this approach are stated. Especially, the disadvantages of existing graph based methods are discussed.

Then, in chapter 3 and 4, a series of novel detection methods for Silent worms, a kind of hit-list worms are presented as follows.

1. Fast detection of hit-list worms based on anomaly connection trees which are constructed attributed to worms propagation (Chapter 3).

2. Distributed detection of hit-list worms based on the prior approach through the cooperation of several IDSes (Chapter 4).

Chapter 3 proposes Anomaly Connection Tree Method (ACTM) that detects worms by tracking large AC trees and VAC trees. This approach takes advantage of the following essential features of most network worm's propagation bahavior.

1. Network worm copies itself onto the infected hosts recursively and the duplicated instance will try to infect other hosts. Then, the worm infection activities can be expressed as a tree structure composed of infected hosts as nodes and Worm Connections (WCs) as edges.

2. When selecting infection targets, the worm does not care about to which hosts and how frequently its infected host has communicated under normal conditions. On the other hand, the most of legitimate connections (LCs) of a host tend to concentrate on a few number of destination hosts. Thus, most parts of WCs are classified as ACs.

Thus, the appearances of tree structures composed of ACs named AC trees strongly indicate the existence of worms. Since small part of WCs are classified as NCs, however, worms connection tress can be divided into small pieces by NCs, and which makes the detection speed slower. To address the problem, ACTM detects areas where AC trees densely appear as VAC trees and uses the area size for detection. ACTM has two phases; learning phase and detection phase. In learning phase, ICs (Internal Connections) are classified in either of NCs or ACs, and then the thresholds of AC trees VAC trees are determined. In the detection phase, ACTM detects worms using parameters specified in the learning phase.

The simulation experiments have shown that ACTM can detect worms before 10% hosts are infected, and is faster than existing graph based methods.

The second approach, which is described in chapter 4, detects the worms in a distributed manner based on ACTM. First, d-ACTM (Distributed ACTM) employs several IDSes named LACDs (Local AC Detectors). Each LACD monitors network activity of its target host and classifies the connections into AC/NC categories. Then, LACDs exchange their local monitoring results to detect AC trees in a distributed manner. A root node of an AC tree estimates the tree size based on the information received from its child nodes. Each LACD maintains a piece of an AC tree in which its target host is included. Then, through the message exchange with other LACDs, a LACD that monitors a host located at the root of the tree estimates the tree size. According to the results of investigation performed by the network manager, each LACD adjusts its detection threshold to satisfy the demanded false positive interval.

Next, to detect VAC trees in addition to AC trees, d-ACTM/VT (d-ACTM with Virtual AC Tree Detection) aggregates trees divided by NCs into VAC trees in a distributed manner.

The simulation experiments have shown that d-ACTM/VT has the same detection performance as ACTM and reduces the amount of messages and logs transmitted in the network to 8% compared to ACTM where a central server has to process all log data received from various traffic capturing points in the network.

In this research, the way to effectively detect hit-list worms in a distributed manner is established. As worms evolve to conceal their existence, more deep inspections will be required. The research's focus that worms propagation constructs tree structures composed of anomaly connections grasps the essential features of worm behavior more deeply compared to existing approaches.

I beleive this reserach will contribute to the improvement of the countermeasures against network worms and other various network based attacks.

Acknowledgement

First of all, I would like to express my heartfelt gratitude toward Professor Ken-ichi Okada and Associate Professor Hiroshi Shigeno for their great supports and advices for six years to obtain this doctoral degree. I would also like to thank Professor Sasase and Associate Professor Khono for reviewing this dissertation and giving me valuable suggestions to improve its quality.

I have been supported and helped by many respectable fellows in Okada&Shigeno Laboratory. Especially, Shintaro Ueda gave me various presents such as the ability of thinking logically and skills in writing technical papers in English. I have enjoyed a great time with JCC members; Hiroaki Ohya, Reina Miyaji, Naohiro Obata, Yusuke Azuma, Shinya Tahara, Taro Inaba and Seiji Shibaguchi. I thank to all super-users who have keenly managed the lab. network and machines. Also, I received much idea and thought from Dr. Inoue, Dr. Egi and Akihiro Miyata.

Finally, from the bottom of my heart, I would like to thank my family for supporting me for many years. Without their dedicated help and understanding, I couldn't accomplish this work.

Thank you all for this precious days.
12 January 2008.

Reference

[1] Zesheng Chen, Lixin Gao, and Kevin Kwiat. Modeling the spread of active worms. In *Proc. of IEEE INFOCOM 2003*, 2003.

[2] S. Staniford-Chen, S. Cheung, R. Crawford, M. Dilger, J. Frank, J. Hoagland, K. Levitt, C. Wee, R. Yip, and D. Zerkle. GrIDS: A Graph-Based Intrusion Detection System for Large Networks. In *Proc. of the 19th National Information Systems Security Conference*, pages 361–370, 1996.

[3] Peter Szor. *The Art of Computer Virus Research and Defense.* Addison Wesley, 2005.

[4] Nicholas Weaver, Vern Paxson, Stuart Staniford, and Robert Cunningham. A taxonomy of computer worms. In *Proc. of The First ACM Workshop on Rapid Malcode 2003*, 2003.

[5] Stuart Staniford, Vern Paxson, and Nicholas Weaver. How to own the internet in your spare time. In *Proc. of the 11th USENIX Security Symposium*, 2002.

[6] Mark Eichin and Jon Rochlis. With microscope and tweezers: An analysis of the internet virus of november 1988. In *Proc. of IEEE Computer Society Symposium on Security and Privacy*, 1989.

[7] Justin Ma, Geoffrey M.Voelker, and Stefan Savage. Self-stopping worms. In *Proc. of the 2005 ACM Workshop on Rapid Malcode*, pages 12–21, 2005.

[8] Zesheng Chen and Chuanyi Ji. A self-learning worm using importance scanning. In *Proc. of the 2005 ACM Workshop on Rapid Malcode*, pages 22–29, 2005.

[9] M.Williamson. Throttling viruses: Restricting propagation to defeat malicious mobile code. In *Technical Report HPL-2002-172*, 2002.

[10] Jaeyeon Jung, Vern Paxson, Arthur, W. Berge, and Hari Balakrishnan. Fast portscan detection using sequential hypothesis testing. In *Proc. of the IEEE Symposium on Security and Privacy*, 2004.

[11] Stuart Staniford, David Moore, Vern Paxson, and Nicholas Weaver. The top speed of flash worms. In *Proc. of ACM 2004 Workshop on Rapid Malcode*, pages 33–42, 2004.

[12] Steve Bellovin, Bill Cheswick, and Angelos Keromytis. worm propagation strategies in an ipv6 internet. In *LOGIN: VOL.31, NO.1*, pages 71–76, 2006.

[13] C.Channon and D.Moore. The spread of the witty worm. *IEEE Security and Privacy 2(4)*, 2(4):46–50, 2004.

[14] Niels Provos, Joe McClain, and Ke Wang. Search worms. In *Proc. of the 4th ACM workshop on Recurring malcode*, pages 1–8, 2006.

[15] Thomas Toth and Christopher Krugel. Connection-history based anomaly detection. In *Proc. of the 3rd IEEE Information Assurance Workshop*, 2002.

[16] Nobutaka Kawaguchi, Yusuke Aazuma, Shintaro Ueda, Hiroshi Shigeno, and kenichi Okada. ACTM: Anomaly Connection Tree Method to detect Silent Worms. In *Proc. of IEEE 20th International Conference on Advanced Information Networking and Applications*, volume 1, pages 901–906, 2006.

[17] Nobutaka Kawaguchi, Hiroshi Shigeno, and kenichi Okada. Detection of Silent Worms using Anomaly Connection Tree. In *Proc. f IEEE 21st International Conference on Advanced Information Networking and Applications*, volume 1, pages 412–419, 2007.

[18] Nobutaka Kawaguchi, Hiroshi Shigeno, Shintaro Ueda, Hidekazu Shiozawa, and Kenichi Okada. ACTM: Anomaly Connection Tree Method for Detection of Silent Worms. *IPSJ Journal*, 48(2):614–643, 2007.

[19] Nobutaka Kawaguchi, Hiroshi Shigeno, and Kenichi Okada. ACTM: Fast Detection Method of Silent Worms using Anomaly Connection Tree. In *IPSJ SIG Technical Report 2006-CSEC-33*, pages 31–36, 2006.

[20] Nobutaka Kawaguchi, Hiroshi Shigeno, and Kenichi Okada. d-ACTM/VT: A Distributed Virtual AC Tree Detection Method. *IPSJ Journal*, 49(2), 2008.

[21] Nobutaka Kawaguchi, Hiroshi Shigeno, and Kenichi Okada. d-ACTM: Distributed Anomaly Connection Tree Method to detect Silent Worms. In *Proc. of 26th IEEE International Performance, Computing and Communications Conference (Malware'07 Track)*, 2007.

[22] Hiroshi Shigeno, Nobutaka Kawaguchi, and Kenichi Okada. A Distributed Worm Detection Method based on ACTM. In *IPSJ SIG Technical Report 2007-DPS-130, 2007-CSEC-36*, pages 201–206, 2007.

[23] Sumeet Singh, Cristian Estan, George Varghese, and Stefan Savage. The early bird system for real-time detection of unknown worms. In *Proc of ACM HotNets 2003*, 2003.

[24] H.-A. Kim and B. Karp. Autograph: Toward automated, distributed worm signature detection. In *Proc. of the 13th Usenix Security Symposium*, 2004.

[25] William Aiello, Chuck Kalmanek, Patrick McDaniel, Subhabrata Sen, Oliver Spatscheck, and Jacobus Van der Merwe. Analysis of communities of interest in data networks. In *Proc. of Passive and Active Measurement Workshop 2005*, 2005.

[26] Ruoming Pang, Mark Allman, Mike Bennett, Jason Lee, Vern Paxson, and Brian Tierney. A first look at modern enterprise traffic. In *Proc. of the Internet Measurement Conference 05*, 2005.

[27] Y. Al-Hammadi and C. Leckie. Anomaly detection for internet worms. In *Proc. of 9th IEEE/IFIP IEEE International Symposium on Integrated Network Management*, pages 133–146, 2006.

[28] Daniel R.Ellis, John G.Aiken, Kira S.Attwood, and Scott D. tengalia. A behavioral approach to worm detection. In *Proc. of 2004 ACM Workshop on Rapid Malcode*, pages 43–53, 2004.

[29] Danniel R.Ellis, John G.Aiken, Adam M.McLeod, and David R.Keppler. Graph-based worm detection on operational enterprise networks. In *MITRE TECHNICAL REPORT*, 2006.

[30] Christopher kruegel and Thomas Toth. Distributed pattern detection for intrusion detection. In *Proc. of Network and Distributed System Security Symposium 2002*, 2002.

[31] C.C.Zou, Weibo Gong, and Don Towsley. Code red propagation modeling and analysis. In *Proc. of the 9th ACM Conference on Computer and Communication Security*, 2002.

[32] David Dagon, Cliff Zou, and Wenke Lee. Modeling botnet propagation using time zones. In *Proc. of the 13th Annual Network and Distributed System Security Symposium*, 2006.

[33] Cliff.C.Zou, Don Towsley, and Weibo Gong. On the performance of internet worm scanning strategies. *International Journal on Performance Evaluation*, 63(7):700–723, 2006.

[34] eeye digital security, blaster worm analysis. http://research.eeye.com/html/advisories/published/AL20030811.html. visited at Oct. 24th 2007.

[35] David moore, Vern Paxon, Stefan Savage, Colleen Shannon, Stuart Staniford, and Nicholas Weaver. Inside the slammer worm. In *IEEE Magazine of Security and Privacy*, pages 33–39, 2003.

[36] Cynthia Wong, Stan Bielski, Jonathan M.McCune, and Chenxi Wang. A study of mass-mailing worms. In *Proc. of the 2004 ACM workshop on Rapid malcode*, pages 1–10, 2004.

[37] C.C.Zou, D.Towsley, and Weibo Gong. Email worm modeling and defense. In *Proc. of 13th International Conference on Computer Communications and Networks*, pages 409–414, 2004.

[38] K.Ishibashi, T.Toyono, K.Toyama, M.Ishino, and H.Ohshima nad I.Mizukoshi. Detecting mass-mailing worm infected hosts by mining dns traffic data. In *Proc. of the 2005 ACM SIGCOMM workshop on Mining network data*, pages 159–164, 2005.

[39] D. Whyte, P.C. van Oorschot, and E. Kranakis. Addressing malicious smtp-based mass-mailing activity within an enterprise network. In *Proc. of Annual Computer Security Applications Conference*, 2006.

[40] Jintao Xiong. ACT: Attachment Chain Tracing Scheme for Email Virus Detection and Control. In *Proc. of the 2004 ACM workshop on Rapid malcode*, pages 11–22, 2004.

[41] C.-T Huang, N.L.Johnson, J.Janies, and A.X.Liu. On capturing and containing e-mail worms. In *Proc. of the 25th IEEE International Performance, Computing and Communications Conference*, page 8, 2006.

[42] C.C.Zou, Weibo Gong, and Don Towsley. Feedback email worm defense system for enterprise networks. In *Umass ECE Technical Report TR-04-CSE-05*, 2004.

[43] Mohammad Mannan and Paul C.van Oorschot. On instant messaging worms. analysis and countermeasures. In *Proc. of the 2005 ACM Workshop on Rapid Malcode*, pages 2–11, 2005.

[44] Zhijun Liu, Guoqiang Shu, Na Li, and David Lee. Defending against instant messaging worms. In *Proc. of IEEE Globecom 2006*, pages 1–6, 2006.

[45] C.C.Zou, D.Towsley, Gong Weibo, and S.Cai. Routing worm: a fast selective attack worm based on ip address information. In *Proc. of Workshop on Principles of Advanced and Distributed Simulation, 2005*, pages 199–206, 2005.

[46] Kannan J. Implications of peer-to-peer networks on worm attacks and defenses. In *California:CS294-4 Project*, 2003.

[47] Wu J, Vangala S, Gao L, and Kwiat K. An effective architecture and algorithm for detecting worms with various scan techniques. In *Proc. of the 11th Symposium on Network and Distributed Systems Security*, pages 143–156, 2004.

[48] Wei Yu, Xun Wang, Prasad Calyam, Dong Xuan, and Wei Zhao. On detecting camouflaging worm. In *Proc. of ACSAC 2006*, 2006.

[49] Nicholas Weaver Dan Ellis Stuart Staniford and Vern Paxson. Worms vs. perimeters: The case for hard-lans. In *In Proc. of 12th Annual IEEE Symposium on High Perfomance Interconnects*, pages 70–76, 2004.

[50] C.Onwubiko, A.P.Lenaghan, and L.Hebbes. An improved worm mitigation model for evaluating the spread of aggressive network worms. In *Proc of IEEE The International Conference on Computer as a Tool 2005*, pages 1710–1713, 2005.

[51] C.C.Zou, WEibo Gong, and Don Towsley. Worm propagation modeling and analysis under dynamic quarantine defense. In *Proc. of the 2003 ACM Workshop on Rapid malcode*, pages 51–60, 2003.

[52] Wei yu, Sriam Chellappan, Xun Wang, and Dong Xuan. On defending peer-to-peer system based active worm attacks. In *Proc. of IEEE Globecom 2005*, pages 1757–1761, 2005.

[53] W.Yu, C.Boyer, and D.Xuan. Analyzing impacts of peer-to-peer systems on propagation of active worm attacks. In *Technical Report, Computer Science Dept*, 2004.

[54] Wei Yu. Analyze the worm-based attack in large scale p2p networks. In *Procc of the Eighth IEEE International Symposium on High Assurance Systems Engineering*, 2004.

[55] R.Thommes and M.Coates. Epidemiological modelling of peer-to-peer viruses and pollution. In *Proc. of The 25th IEEE INFOCOM*, pages 1–12, 2006.

[56] Guanling Chen and Robert S.Gray. Simulating non-scanning worms on peer-to-peer networks. In *Proc. of ACM 1st International Conference on Scalable Information systems*, page 29, 2006.

[57] James W.Mickens and Brian D.Noble. Modeling epidemic spreading in mobile environments. In *Proc. of the 4th ACM workshop on Wireless security*, pages 77–86, 2005.

[58] Jing Su, Kelvin K. W. Chan, Andrew G. Miklas, Kenneth Po, Ali Akhavan, Stefan Saroiu, Eyal de Lara, and Ashvin Goel. A preliminary investigation in a bluetooth environment. In *Proc. of the 4th ACM workshop on Recurring malcode*, pages 9–16, 2006.

[59] Stuart E.Schechter, Jaeyeon Jung, WIll Stockwell, and Cynthia McLain. Inoculating ssh against address-harvesting worms. In *In Proc. of The Network and Distributed System Security Symposium 2006*, 2006.

[60] Syed Al Khayam and Hayder Radha. A topologically-aware worm propagation model for wireless sensor networks. In *Proc. of IEEE ICDCDS International Workshop on Security in Distributed Computing Systems*, 2005.

[61] M.Nekovee. Modeling the spread of worm epidemics in vehicular ad hoc networks. In *Proc.of the IEEE 63rd Vehicular Technology Conference*, pages 841–845, 2006.

[62] David Whyte, Evangelos Kranakis, and P.C. van Oorschot. Dns-based detection of scanning worms in an enterprise network. In *In Proc. of The Network and Distributed System Security Symposium 2005*, 2005.

[63] M.Patrick Collins and Michael K.Reiter. Hit-list worm detection and bot identification in large networks using protocol graphs. In *Proc. 10th International Symposium on Recent Advances of Intrusion Detection 2007*, pages 276–295, 2007.

[64] L.Zhou, L.Zhang, F.McSherry, N.Immorlica, M.Costa, and S.Chien. A first look at peer-to-peer worms: Threats and defenses. In *Proc. of 4th International Workshop on Peer-To-peer Systems*, 2005.

[65] Nessus, the network vulnerability scanner.
http://www.nessus.org/. visited at Oct. 28th 2007.

[66] Adam J.O'Donnell and Harish Sethu. On achieving software diversity for improved network security using distributed coloring algorithms. In *Proc. of ACM CCS'04*, 2004.

[67] Adam J.O'Donnel and Harish Sethu. Software diversity as a defense against viral propagation: Models and simulations. In *Proc. of the IEEE workshop on Principles of Advanced and Distributed Simulation 2005*, pages 247–253, 2005.

[68] S.Antonatos, P.Akritidis, E.P.Markatos, and K.G.Anagnostakis. Defending against hitlist worms using network address space randomization. In *Proc. of the 2005 ACM Workshop on Rapid Malcode*, pages 30–40, 2005.

[69] K.G.Anagonstakis and S.Antonatos. Tao: Protecting against hitlist worms using transparent address obfuscation. In *Proc. of the 10th IFIP Open Conference on Communications and Multimedia Security*, 2006.

[70] R.Janakiraman, M.Waldvogel, and Q.Zhang. Indra: A peer-to-peer approach to netwrok intrusion detection and prevention. In *Proc. of IEEE WEITCE 2003*, 2003.

[71] D. Nojiri, J. Rowe, and K. Levitt. Cooperative response strategies for large scale attack mitigation. In *Proc. of DARPA Information Survivability Conference and Exposition 2003*, pages 293–302, 2003.

[72] Jayanthkumar Kannan, Lakshminarayanan Subramanian, Ion Stoica, and Randy H. Katz. Analyzing cooperative containment of fast scanning worms. In *Proc. of The 1st Steps to Reducing Unwanted Traffic on the Internet Workshop*, 2005.

[73] K.G Anagnostakis, M.B. Greenwald, S. Ioannidis, A.D Keromytis, and Dekai Li. A cooperative immunization system for an untrusting internet. In *Proc. of The 11th IEEE International Conference on Networks*, pages 403–408, 2003.

[74] Phillip Porras, Linda Briesemeister, Keith Skinner, Karl Levitt, Jeff Rowe, and Yu-Cheng Allen Ting. A hybrid quarantine defense. In *Proc. of the 2004 ACM Workshop on Rapid Malcode (WORM)*, pages 73–82, 2004.

[75] Senthilkumar G. Cheetancheri, John Mark Agosta, Denver H. Dash, Karl N. Levitt, Jeff Rowe, and Eve M. Schooler. A distributed host-based worm detection system. In *Proc. of the 2006 SIGCOMM workshop on Large-scale attack defense*, pages 107–113, 2006.

[76] Vinod Yegneswaran, Paul Barford, and Somesh Jha. Global intrusion detection in the domino overlay system. In *Proc. of The 11th Annual Network and Distributed System Security Symposium*, 2004.

[77] Locasto M, Parekh J, Stolfo S, Keromytis A, Malkin T, and Misra V. Collaborative distributed intrusion detection. In *CU Tech REport CUCS-012-04*, 2004.

[78] Yinglian Xie, Vyas Sekar, David A. Maltz, Michael K. Reiter, and Hui Zhang. Worm origin identification using random moonwalks. In *Proc. of the 2005 IEEE Symposium on Security and Privacy*, pages 242–256, 2005.

[79] Vyas Sekar, Yinglian Xie, David A, Maltz Michael, K. Reiter, and Hui Zhang. Toward a framework for internet forensic analysis. In *Proc. of ACM HotNets-III*, 2004.

[80] Vyas Sekar, Yinglian Xie, Michael K.Reiter, and Hui Zhang. Is host-based anomaly detection+temporal correlation=worm causality? In *CMU-CS-07-112*, 2007.

[81] Yinglian Xie, Vyas Sekar, Michael K.Reiter, and Hui Zhang. Forensic analysis for epidemic attacks in federated networks. In *Proc. of IEEE ICNP 2006*, pages 44–53, 2006.

[82] Yinglian Xied and. Hyang-Ah Kim, David OHallaron, Michael K. Reiter, and Hui Zhang. Seurat: A pointillist approach to anomaly detection. In *Proc. of the 7th International Symposium on Recent Advances in Intrusion Detection*, 2004.

[83] David J. Malan and Michael D. Smith. Host-based detection of worms through peer-to-peer cooperation. In *Proc. of the 2005 ACM Workshop on Rapid Malcode*, pages 72–80, 2005.

[84] Snort.
http://www.snort.org/. visited at Oct. 31st 2007.

[85] D.Moore, C.Shannon, G.Voelker, and S.Savage. Internet quarantine: Requirements for containing self-propagating code. In *Proc. of IEEE INFOCOM*, 2003.

[86] Georgios Portokalidis and Herbert Bos. Sweetbait: Zero-hour worm detection and containment using honeypots. In *Technical Report IR-CS-015*, 2005.

[87] Christian Kreibich and Jon Crowcroft. Honeycomb: creating intrusion detection signatures using honeypots. In *Proc. of ACM SIGCOMM Computer Communication Review*, pages 51–56, 2004.

[88] Yegneswaran Vinod, Barford Paul, and Dave Plonka. On the design and use of internet sinks for network abuse monitoring. In *Proc. of 7th International Symposium on Recent Advances in Intrusion Detection*, 2004.

[89] Sumeet Singh, Cristian Estan, George Varghese, and Stefan Savage. Automated worm fingerprinting. In *Proc of 6th USENIX Symposium on Operating Systems Design and Implementation*, 2004.

[90] P.Akritidis, K.Anagnostakis, and E.P.Markatos. Efficient content-based detection of zero-day worms. In *Proc of IEEE ICC 2005*, volume 2, pages 837–843, 2005.

[91] M.Cai, R.Zou, K.Hang, C.Papadopoulos, and S.Song. Wormshield: Collaborative worm signature detection using distributed aggregation trees. In *Technical Report, USC Internet and Grid Computing Lab*, 2005.

[92] K. Hwang, Y.K. Kwok, S. Song, M. Cai, Yu Chen, R. Zhou, Ying Chen, and X. Lou. Gridsec: Trusted grid computing with security binding and self-defense against network worms and ddos attacks. In *Proc. of International Workshop on Grid Computing Security and Resource Management (GSRM-2005)*, pages 187–195, 2005.

[93] Min Cai, Kai Hwang, Yu-Kwong Kwok, Shanshan Song, and Yu Chen. Collaborative internet worm containment. In *IEEE Security & Privacy Magazine*, volume 3, pages 25–33, 2005.

[94] Christopher Kruegel, Engin Kirda, Darren Mutz, William Robertson, and Giovanni Vigna. Polymorphic worm detection using structural information of executables. In *Proc. of the 8th Symposium on Recent Advances in Intrusion Detection*, 2005.

[95] J.Newsome and D.Song. Polygraph: automatically generating signatures for polymorphic worms. In *Proc. of the IEEE Symposium on Security and Privacy 2005*, pages 226–241, 2005.

[96] Zhichun Li, Manan Sanghi, Yan Chen, Ming-Yang Kao, and Brian Chaves. Hamsa: Fast signature generation for zero-day polymorphic worms with provable attack resilience. In *Proc of IEEE Symposium on Security and Privacy '06*, 2006.

[97] Y.Tang and S.Chen. Defending against internet worms: a signature-based approach. In *Proc. of the 24th IEEE INFOCOM*, volume 2, pages 1384–1394, 2005.

[98] Roberto Perdisci, David Dagon, Wenke Lee, Prahlad Fogla, and Monirul Shartif. Misleading worm signature generators using deliberate noise injection. In *Proc. of IEEE Symposium on Security and Privacy '06*, 2006.

[99] Ke Wang and Salvatore J. Stolfo. Anomalous payload-based network intrusion detection. In *Proc. of the 7th International Symposium on Recent Advances in Intrusion Detection*, 2004.

[100] Ke Wang, Gabriela Cretu, and Salvatore J. Stolfo. Anomalous payload-based worm detection and signature generation. In *Proc. of the 8th International Symposium on Recent Advances in Intrusion Detection*, 2005.

[101] Ke Wang, Janak J.Parekh, and Salvatore J.Stolfo. Anagram: A content anomaly detector resistant to mimicry attack. In *Proc. of the 9th International Symposium on Recent Advances in Intrusion Detection*, 2006.

[102] Helen J.Wang, Chuanxiong Guo, Daniel R. Simon, and Alf Zugenmaier. Shield: Vulnerability-driven network filters for preventing known vulnerability exploits. In *Proc. of ACM SIGCOMM'04*, 2004.

[103] Charles Reis, John Dunagan, HelenJ.Wang, Opher Dubrovsky, and Saher Esmeir. Browsershield: Vulnerability-driven filtering of dynamic html. In *Proc. of 7th USENIX Symposium on Operating Systems Design and Implementation*, pages 61–73, 2006.

[104] Ashlesha Joshi, Samuel T.King, George W. Dunlap, and Peter M.Chen. Detecting past and present intrusions through vulnerability-specific predicates. In *Proc. of ACM SOSP'05*, 2005.

[105] Manuel Costa, Jon Crowcroft, Miguel Castro, Antony Rowstorn, Lidong Zhou, Lintao Zhang, and Paul Barham. Vigilante: End-to-end containment of internet worms. In *Proc. of ACM SOSP'05*, 2005.

[106] Zhenkai Liang and R.Sekar. Fast and automated generation of attack signatures: A basis for building self-protecting servers. In *Proc. of ACM CCS'05*, 2005.

[107] Zhenkai Liang and R.Sekar. Automatic synthesis of filters to discard buffer overflow attacks: A step towards realizing self-healing systems. In *Proc. of USENIX Annual Technical Conference 2005*, 2005.

[108] Seth Robertson, Eric V. Siegel, Matt Miller, and Salvatore J. Stolfo. Surveillance detection in high bandwidth environments. In *Proc. of the 2004 DARPA DISCEX III Conference*, 2003.

[109] Shigang Chen and Sanjay Ranka. An internet-worm early warning system. In *Proc. of IEEE Global Communication Conference*, 2004.

[110] George Bakos and Vincent H.Berk. Early detection of internet worm activity by metering icmp destination unreachable messages. In *In Proc. of SPIE 2002*, volume 4708, pages pp.33–42, 2002.

[111] Vincent H. Berk, Robert S. Gray, and George Bakos. Using sensor networks and data fusion for early detection of active worms. In *Proc. of the SPIE Aerosense conference*, 2003.

[112] Guofei Gu, Monirul Sharif, Xinzhou Qinand David Dagon, Wenke Lee, and George Riley. Worm detection, early warning and response based on local victim information. In *Proc. of IEEE 20th Annual Computer Security Application Conference*, pages 136–145, 2004.

[113] Xuxian Jaing and Dongyan Xu. Cybertrap: Detecting and quaranting scanning worms in enterprise networks. In *Department of Computer Science Technical Report*, 2004.

[114] Cliff Changchun Zou, Lixin Gao, Weibo Gong, and don Towsley. Monitoring and early warning for internet worms. In *Proc. of 10th ACM Symposium on Computer and Communication Security*, 2003.

[115] Cliff Changchun Zou, Weibo Gong, Don Towsley, and Lixin Gao. The monitoring and early detection of internet worms. In *Proc. of IEEE/ACM Transactions on Networking*, pages 961–974, 2005.

[116] David Whyte, P.C. van Oorschot, and Evangelos Kranakis. Arp-based detection of scanning worms within an enterprise network. In *Carleton Univ Technical Report*, 2005.

[117] Shingang Chen and Yong Tang. Slowing down internet worms. In *Proc. of the IEEE 24th International Conference on Distributed Computing Systems*, pages 312–319, 2004.

[118] Stuart Schechter, Jaeyeon Jung, and Arthur W. Berger. Fast detection of scanning worm infections. In *Proc. of RAID 2004*, pages 102–124, 2004.

[119] Bo Chen, Fang Bin-Xing, and yun Xiao-Chun. A enw approach for early detection of internet worms based on connection degree. In *Proc. of 2005 International Conference on Machine Learning and Cybernetics*, volume 4, pages 2424–2430, 2005.

[120] Patrick McDaniel, Shubho Sen, Oliver Spatscheck, Jacobus Van der Merwe, Bill Aiello, and Charles Kalmanek. Enterprise security: A community of interest based approach. In *Proc. of The 13th Annual Network and Distributed System Security Symposium*, 2006.

[121] Cynthia Wong, Chenxi Wang, Sawn Song, Stan Bielski, and Gregory R.Ganger. Dynamic quarantine of internet worms. In *Proc. of the 2004 International Conference on Dependable Systems and Networks*, pages 73–73, 2004.

[122] Naisir aJamil and Thomas M.Chen. Effectiveness of rate control in slowing down worm epidemics. In *Proc. of IEEE Globecom 2006*, pages 1–6, 2006.

[123] Sanjeev Dwivedi. Feedback-based worm containment. In *Proc. of 20th International Conference on Advanced Information Networking and Applications*, volume 1, pages 883–888, 2006.

[124] T.M.Chen and N.Jamil. Effectiveness of quarantine in worm epidemics. In *Proc. of IEEE ICC 2006*, volume 5, pages 2142–2147, 2006.

[125] Vyas Sekar, Yinglian Xie, Michael K. Reiter, and Hui Zhang. A multi-resolution approach for worm detection and containment. In *Proc. of the International Conference on Dependable Systems and Networks 06*, pages 189–198, 2006.

[126] Sarah Sellke. Modeling and automated containment of worms. In *Proc. of the 2005 International Conference on Dependable Systems and Networks*, 2005.

[127] Songging Chen, Xinyuan Wang, Lei Liu, and Xinwen Zhang. Worm terminator: an effective containment of unknown and polymorphic fast spreading worms. In *Proc. of the 2006 ACM/IEEE symposium on Architecture for networking and communication systems*, pages 173–182, 2006.

[128] Avinash Sridharan Ye and T.Suparatik Bhattacharyya. Connectionless port scan detection on the backbone. In *Proc. of 25th IEEE International Conference on Performance, Computing and Communications Conference*, pages 10–10, 2006.

[129] Sanguk Noh, Cheolho Lee, Keywon Ryu, Kyunghee Choi, and Gihyun Jung. Detecting worm propagation using traffic concentration analysis and inductive learning. In *Proc. of IDEAL 2004*, pages 402–408, 2004.

[130] Arno Wagner and Bernhard Plattner. Entropy based worm and anomaly detection in fast ip networks. In *In Proc. of 14th IEEE International Workshop on Enabling Technologies: Infrastructures for Collaborative Enterprises*, pages 172–177, 2005.

[131] Xuan Chen and John Heidemann. Detecting, early worm propagation through packet matching. In *Technical Report ISI-TR-2004-585, USC/Information Sciences Institute*, 2004.

[132] Stuart Staniford. Containment of scanning worms in enterprise networks. In *Journal of Computer Security*, 2004.

[133] Wang Ping and Fang Bin-Xing Yun Xiao-Chun. A user habit based approach to detect and quarantine worms. In *Proc. of IEEE ICC 2006*, pages 2148–2152, 2006.

[134] Jin ho Kim, Hyogon Kim, and Saewoong Bahk. On the effectiveness of service registration worm defense. In *Proc. of IEEE Globecom 2006*, pages 1–6, 2006.

[135] J.Chan, C.Leckie, and T.peng. Hit-list worm detection using source ip address history. In *Proc. of Australian Telecommunication and Applications Conference*, 2006.

[136] Chin-Tser Huang, Nathan L. Johnson, and Jeff Janies. On capturing and containing e-mail worms. In *Proc. of IEEE IPCCC 2006*, 2006.

[137] Taro Inaba, Nobutaka Kawaguchi, Shinya Tahara, Hiroshi Shigeno, and Kenichi Okada. Early containment of worms using dummy addresses and connection trace back. In *Proc. of the 14th International Conference on Parallel and Distributed Systems*, 2007.

Achievement List

Journal Papers related to this Thesis

1. Nobutaka Kawaguchi, Hiroshi Shigeno, Shintaro Ueda, Hidekazu Shiozawa, Kenichi Okada, "ACTM: Anomaly Connection Tree Method for Detection of Silent Worms", IPSJ Journal, Vol.48, No.2, pp.614-624, 2007.

2. Nobutaka Kawaguchi, Hiroshi Shigeno, Kenichi Okada, "d-ACTM/VT: A Distributed Virtual AC Tree Detection Method", IPSJ Journal, Vol.49, No.2, 2008.

International Conferences related to this Thesis

1. Nobutaka Kawaguchi, Yusuke Azuma, Shintaro Ueda, Hiroshi Shigeno, Kenichi Okada, "ACTM: Anomaly Connection Tree Method to Detect Silent Worms", In Proc. of The IEEE 20th International Conference on Advanced Information Networking and Applications, Vol.1, pp.901-906, April 2006.

2. Nobutaka Kawaguchi, Hiroshi Shigeno, Kenichi Okada, "d-ACTM: Distributed Anomaly Connection Tree Method to detect Silent Worms", In Proc. of 2nd International Swarm Intelligence & Other Forms of Malware Workshop, pp.510-517, April 2007.

3. Nobutaka Kawaguchi, Hiroshi Shigeno, Kenichi Okada, "Detection of Silent Worms using Anomaly Connection Tree", In Proc. of The IEEE 21st International Conference on Advanced Information Networking and Applications, pp.412-419, May 2007.

4. Nobutaka Kawaguchi, Hiroshi Shigeno, Kenichi Okada, "A Distributed Detection of Hit-list Worms", In Proc. of The IEEE 2008 International Conference on Communications, May 2008.

Other Journal Papers

1. Shintaro Ueda, Shuichi Eto, Nobutaka Kawaguchi, Ryuya Uda, Hiroshi Shigeno, Kenichi Okada, "Real-time Stream Authentication Scheme for IP Telephony", IPSJ Journal, Vol.45, No.2, pp.605-613, 2004.

2. Hiroaki Ohya, Reina Miyaji, Nobutaka Kawaguchi, Hiroshi Shigeno, Kenichi Okada, "A Technique to Reduce False Positives of Network IDS with Machine Learning", IPSJ Journal, Vol.45, No.8, pp.2105-2122, 2004.

3. Shintaro Ueda, Shin-ichiro Kaneko, Nobutaka Kawaguchi, Hiroshi Shigeno, Kenichi Okada, "A Real-Time Stream Authentication Scheme for Video Streams", IPSJ Journal, Vol.47, No.2, pp.415-425, 2006.

Other International Conferences

1. Shintaro Ueda, Nobutaka Kawaguchi, Hiroshi Shigeno, Kenichi Okada, "Stream Authentication Scheme for the use over the IP Telephony", The 18th International Conference on Advanced Information Networking and Applications(AINA 2004), pp.164-169, March 2004.

2. Nobutaka Kawaguchi, Shintaro Ueda, Naohiro Obata, Reina Miyaji, Shinichiro Kaneko, Hiroshi Shigeno, Kenichi Okada, "A Secure Logging Scheme for Forensic Computing", 5th Annual IEEE Information Assurance Workshop, pp.386-393, June 2004.

3. Nobutaka Kawaguchi, Shintaro Ueda, Naohiro Obata, Hiroshi Shigeno, Kenichi Okada, "An Image Authentication Scheme considering Privacy -A First Step towards a Surveillance Camera Authentication", The First International Workshop on Information Networking and Applications(INA'2005), pp.253-256, March 2005.

4. Nobutaka Kawaguchi, Naohiro Obata, Shintaro Ueda, Yusuke Azuma, Hiroshi Shigeno, Kenichi Okada, "Efficient log authentication for Forensic Computing", IEEE 6th Information Assurance Workshop, pp.215-223, June 2005.

5. Shintaro Ueda, Shin-ichiro Kaneko, Nobutaka Kawaguchi, Hiroshi Shigeno, Kenichi Okada, "Authenticating Video Streams", The IEEE 20th International Conference on Advanced Information Networking and Applications(AINA 2006), Vol.1, pp.863-868, April 2006.

6. Nobutaka Kawaguchi, Yusuke Azuma, Shinya Tahara, Hidekazu Shiozawa, Hiroshi Shigeno, Kenichi Okada, "CMSF: Cooperative Mobile Network Security Information Distribution Framework", in Proc. of The 3rd International Conference on Mobile Computing and Ubiquitous Networking, pp.99-106, Oct. 2006.

7. Shinya Tahara, Nobutaka Kawaguchi, Taro Inaba, Hidekazu Shiozawa, Hiroshi Shigeno, Kenichi Okada, "MSP-system: Mobile Secure Passport System to detect Malicious Users", In Proc. of 8th IEEE SMC Information Assurance Workshop, pp.277-283, June. 2007.

8. Taro Inaba, Nobutaka Kawaguchi, Sinya Tahara, Hiroshi Shigeno, Kenichi Okada, "Early Containment of Worms Using Dummy Addresses and Connection Trace Back", In Proc.

of The 13th International Conference on Pararell and Distributed Systems (ICPADS), Dec. 2007.

9. Taro Inaba, Shinya Tahara, Nobutaka Kawaguchi, Seiji Shibaguchi, Hidekazu Shiozawa, Kenichi Okada, “Visualization of Worm Path Identification”, In Proc. of The Fourth Anuual IFIP WG 11.9 International Conference on Digital Forensics, Jan. 2008.

Short Paper

1. Yusuke Azuma, Naohiro Obata, Nobutaka Kawaguchi, Hidekazu Shiozawa, Hiroshi Shigeno, Kenichi Okada, “Providing Security Information of Mobile Networks using Personal IDS”, Proc. of 4th Forum on Informatoin Technology, pp.281-282, Sep. 2005.

Technical Reports

1. Reina Miyaji, Hiroaki Ohya, Nobutaka Kawaguchi, Hiroshi Shigeno, Kenichi Okada, “A Proposal for Technique to Reduce False Positive of Network IDS with Machine Learning”, IPSJ SIG Technical Report 2003-CSEC-21, pp.53-58, May 2003.

2. Nobutaka Kawaguchi, Naohiro Obata, Reina Miyaji, Shintaro ueda, Hiroshi Shigeno, Kenichi Okada, “A Proposal of P2P Credential Framework”, IPSJ SIG Technical Report 2003-CSEC-22, pp.243-248, July 2003.

3. Shinichiro Kaneko, Shintaro Ueda, Nobutaka Kawaguchi, Hiroshi Shigeno, Kenichi Okada, “Stream Authentication Schem using IDA for Motion Pictures”, Computer Security Symposium 2003, pp.361-366, Nov. 2003.

4. Nobutaka Kawaguchi, Miyaji Reina, Hiroshi Shigeno, Kenichi Okada, “A Proposal of Confidential File Viewing System”, IPSJ 11th Multimedia Communication and Distributed Processing System Workshop, pp.197-202, Dec, 2003.

5. Shintaro ueda, Nobutaka Kawaguchi, Hiroshi Shigeno, Kenichi Okada, “Secure Efficient Logging Architecture for Forensic Computing”, IPSJ SIG Technical Report 2003-CSEC-23, pp.7-12, Dec. 2003.

6. Naohiro Obata, Miyaji Reina, Nobutaka Kawaguchi, Hiroshi Shigeno Kenichi Okada, “Worm Propagation Simulation Considering Online Hosts”, IPSJ SIG Technical Report 2004-DPS-117 2004-CSEC-24, pp75-80, March 2004.

7. Shinichiro Kaneko, Shintaro Ueda, Nobutaka Kawaguchi, Hiroshi Shigeno, Kenichi Okada, “Authentication Information Distributed Scheme using Forward Error Correction”, IPSJ SIG Technical Report 2004-DPS-118, pp.43-48, June 2004.

8. Naohiro Obata, Nobutaka Kawaguchi, Hidekazu Shiozawa, Hiroshi Shigeno, Kenichi Okada, “Worm Propagation Simulation Considering Human Action”, IPSJ SIG Technical Report 2004-GN-53, pp7-12, Sep. 2004.

9. Shinichiro Kaneko, Shintaro Ueda, Nobutaka Kawaguchi, Takeshi Ogino, Hiroshi Shigeno, Kenichi Okada, “Proposal of Real-time Stream Authentication Scheme for Motion Pictures”, IPSJ SIG Technical Report 2005-DPS-122 2005-CSEC-28, pp.211-216, March 2005.

10. Naohiro Obata, Nobutaka Kawaguchi, Yusuke Azuma, Hiroshi Shigeno, Kenichi Okada, “Effecient log authentication for Forensic Computing”, IPSJ SIG Technical Report 2005-DPS-123, pp.31-36, June 2005.

11. Yusuke Azuma, Nobutaka Kawaguchi, Naohiro Obata, Hidekazu Shiozawa, Hiroshi Shigeno, Kenichi Okada, “Security Information Provider Service of mobile Networks using Personal IDS”, IPSJ SIG Technical Report 2005-DPS-125 2005-EIP-29 pp.31-36, Nov. 2005.

12. Nobutaka Kawaguchi, Hiroshi Shigeno, Kenichi Okada, “ACTM: Fast Detection Method of Silent Worms using Anomaly Connection Tree”, IPSJ SIG Technial Report 2006-CSEC-33, pp.31-36, May 2006.

13. Yusuke Azuma, Nobutaka Kawaguchi, Hiroshi Shigeno, Kenichi Okada, “A System Call Log Analysis to Detect Malicious Packets”, IPSJ DICOMO 2006, pp.737-740, July 2006.

14. Shinya Tahara, Yusuke Azuma, Nobutaka Kawaguchi, Hidekazu Shiozawa, Kenichi Okada, “Visualizing Security Information of Mobile Network considering Geographical Location”, IPSJ SIG Technical Report 2006-DPS-128 2006-GN-61 2006-EIP-33, pp.23-28, Sep. 2006.

15. Nobutaka Kawaguchi, Yusuke Azuma, Shinya Tahara, Hidekazu Shiozawa, Hiroshi Shigeno, Kenichi Okada, “Cooperative Mobile Network Security Information Distribution Framework”, IPSJ 14th Multimedia Communidaion and Distributed Processing System Workshop, pp.1-6, Nov. 2006.

16. Hiroshi Shigeno, Nobutaka Kawaguchi, Kenichi Okada, “A Distributed worm Detection Method based on ACTM”, IPSJ SIG Technical Report, 2007-DPS-130, 2007-CSEC-36, pp.201-pp.206, March 2007.

17. Taro Inaba, Nobutaka Kawaguchi, Shinya Tahara, Yusuke Azuma, Hiroshi Shigeno, Kenichi Okada, “Containment of Worms with Dummy Addresses”, pp267-272, IPSJ SIG Technical Report, 2007-DPS-130, 2007-CSEC-36, pp.267-272, March 2007.

18. Shinya Tahara, Nobutaka Kawaguchi, Taro Inaba, Hidekazu Shiozawa, Hiroshi Shigeno, Kenichi Okada, “Cooperative Detection of Malicious Mobile Users using Network Activity History”, IPSJ DICOMO 2007, pp.1670-1677, July 2007.

19. Taro Inaba, Nobutaka Kawaguchi, Shinya Tahara, Hiroshi Shigeno, Kenichi Okada, “Worm Containment with Dummy Addresses and Connection Trace Back”, IPSJ 14th Multimedia Communidaion and Distributed Processing System Workshop, Nov. 2007.

20. Seiji Shibaguchi, Taro Inaba, Nobutaka Kawaguchi, Shinya Tahara, Hidekazu Shiozawa, Kenichi Okada, “Visualization of File Transmission Route for Digital Forensics”, IPSJ SIG Technical Report, 2008-GN-66, Jan. 2008.

Oral Presentations

1. Nobutaka Kawaguchi, Reina Miyaji, Hiroaki Ohya, Hiroshi Shigeno, Kenichi Okada, “Browsing Management of Confidential Files Using Java Card”, IPSJ 65th National Convention, Vol.4, pp.459-460, March 2005.

2. Nobutaka Kawaguchi, Reina Miyaji, Shintaro ueda, Hiroshi Shigeno, Kenichi Okada, “Management of Browsing Confidential Documuments by Location Authentication”, First Forum on Information Technoogy, pp.229-230, Sep. 2003.

3. Shintaro ueda, Shuichi Eto, Nobutaka Kawaguchi, Ryuya Uda, Hiroshi Shigeno, Kenichi Okada, “Voice Stream Authentication Method Proposal Assuming IP Phone”, First Forum on Information Technology, pp.353-354, Sep. 2003.

4. Yusuke Azuma, Naohiro Obata, Nobutaka Kawaguchi, Hidekazu Shiozawa, Hiroshi Shigeno, Kenichi Okada, Security Information Provider Service for Mobile Network, IPSJ 67th National Convention, Vol.3, pp295-296, March 2005.

5. Takeshi Ogino, Shin-ichiro Kaneko, Shintaro Ueda, Nobutaka Kawaguchi, Hiroshi shigeno, Kenichi Okada, “Implementing Message Authentication to Real-Time Streaming Applications”, FIT2005, pp.283-284, Sep. 2005.

6. Shinya Tahara, Yusuke Azuma, Naohiro Obata, Nobutaka Kawaguchi, Hiroshi Shigeno, Kenichi Okada, “Visualizing Security Information of Mobile Network Considering Geographics Location”, IPSJ 68th National Convention, vol.3, pp343-344, March 2006.

Award

1. Keio University Nakanishi Incentive Award (received at 2003.3.23), by Bachelor Thesis “Proposal of Secure Browsing Systems of Condifential Files Using Java Card”.

2. IPSJ 65th National Convention, Student Incentive Award (received at 2003.3.27), by “Browsing Management of Confidential Files Using Java Card”.

3. IPSJ 14th Multimedia Communidaion and Distributed Processing System Workshop, Best Conversant Award (received at 2006.12.1).

4. Keio University International Conference Presentation Incentive Award (received at 2007.1.17), by “CMSF: Cooperative Mobile Network Security Information Distribution Framework”.