BEHAVIOR-BASED WORM DETECTION

by

JOHN SHADRACH STAFFORD

A DISSERTATION

Presented to the Department of Computer and Information Science
and the Graduate School of the University of Oregon
in partial fulfillment of the requirements
for the degree of
Doctor of Philosophy

March 2012

DISSERTATION APPROVAL PAGE

Student: John Shadrach Stafford

Title: Behavior-based Worm Detection

This dissertation has been accepted and approved in partial fulfillment of the requirements for the Doctor of Philosophy degree in the Department of Computer and Information Science by:

Dr. Jun Li	Chair
Dr. John Conery	Member
Dr. Chris Wilson	Member
Dr. Yuan Xu	Outside Member

and

Kimberly Andrews Espy	Vice President for Research & Innovation/ Dean of the Graduate School

Original approval signatures are on file with the University of Oregon Graduate School.

Degree awarded March 2012

DISSERTATION ABSTRACT

John Shadrach Stafford

Doctor of Philosophy

Department of Computer and Information Science

March 2012

Title: Behavior-based Worm Detection

The Internet has become a core component of our lives and businesses. Its reliability and availability are of paramount importance. There are many types of malware that impact the availability of the Internet, including network worms, bot-nets, viruses, etc. Detecting such attacks is a critical component of defending against them. This dissertation focuses on detecting and understanding self-propagating network worms, a type of malware with a proven record of disrupting the Internet. According to *Computer Economics*, the Code-Red worm caused more than 2.5 billion dollars in damages, and it was an unsophisticated worm that hit nearly 10 years ago when the Internet was less important than it is now. The recent StuxNet worm is a tremendously more sophisticated worm than Code-Red, and had it been targeted at disrupting the Internet it seems a near certainty that it could have caused significantly more damage than Code-Red. For this reason it is supremely important that we focus on detecting and stopping worms. Many worm detectors have been proposed and are being deployed, but the literature does not clearly indicate which one is best. New worms such as IKEE.B (also known as the iPhone worm) present new challenges to worm detection, raising the question of how effective our worm defenses are.

This dissertation studies the detection of self-propagating network worms with the goal of improving our ability to detect slowly propagating "stealthy" worms. We make the following contributions to the field: (i) we introduce a worm-detector evaluation framework that allows us to easily evaluate a detector's performance across a variety of environments and worm types; (ii) we use this evaluation environment to compare existing worm detectors to determine their strengths and weaknesses; (iii) we examine evasive worms that attempt to avoid detection, measuring how effective they are at remaining undetected and the propagation rate they are able to achieve while doing so; and (iv) we introduce a new worm detector, SWORD2, which provides superior performance at detecting stealthy or evasive worms.

This dissertation includes previously published co-authored material.

CURRICULUM VITAE

NAME OF AUTHOR: John Shadrach Stafford

GRADUATE AND UNDERGRADUATE SCHOOLS ATTENDED:

University of Oregon, Eugene
Carleton College, Northfield, MN

DEGREES AWARDED:

Doctor of Philosophy, Computer and Information Science, 2012, University of Oregon
Master of Science, Computer and Information Science, 2005, University of Oregon
Bachelor of Arts, Computer Science, 1996, Carleton College

AREAS OF SPECIAL INTEREST:

Worm Detection, Network Behavior, Computational Science

PROFESSIONAL EXPERIENCE:

Senior Software Engineer, Palo Alto Software, 2011-current

Senior Software Consultant, Gecko Designs, 2006-2007

Graduate Research Fellow, NetSec Research Lab, University of Oregon, 2005-2008

Graduate Teaching Fellow, Department of Computer and Information Science, University of Oregon, 2004-2005 and 2006-2007

Senior Software Consultant, Redside Solutions, 2003

Senior Software Consultant, Meridian Technology Group, 1999-2003

Internet Application Engineer, MetroOne Telecommunications, 1998-1999

Software Engineer, Integrity Solutions, 1996-1998

GRANTS, AWARDS AND HONORS:

Computer Science Graduate Teaching Fellow of the Year, University of Oregon, 2007

Award of Distinction on Directed Research Project, University of Oregon, 2006

Award of Distinction on Senior Comprehensive Project, Carleton College, 1996

PUBLICATIONS:

S. Stafford, J. Li, "Internet Worm Detection Techniques: A Survey," Technical Report CIS-TR-2012-01, University of Oregon, 2012.

S. Stafford and J. Li, "Behavior-based Worm Detectors Compared," in *Proceedings of the Recent Advances in Intrusion Detection (RAID) Symposium*, pp. 38–57, 2010.

S. Stafford, J. Li, and T. Ehrenkranz, "Enhancing SWORD to Detect Zero-Day-Worm-Infected Hosts," in *SIMULATION*, vol. 83, pp. 199-212, 2007.

S. Stafford, J. Li, and T. Ehrenkranz, "On the Performance of SWORD in Detecting Zero-Day-Worm-Infected Hosts," in *Proceedings of the Symposium on Performance Evaluation of Computer and Telecommunication Systems*, vol. 38.3, pp. 559-566, 2006.

S. Stafford, T. Ehrenkranz, and J. Li, "Detecting Zero-Day Self-Propagating Internet Worms Based on Their Fundamental Behavior," in *Proceedings of the USENIX Security Symposium*, Poster, 2006.

S. Stafford, J. Li, T. Ehrenkranz, and P. Knickerbocker, "GLOWS: A High-fidelity Worm Simulator," Technical Report CIS-TR-2006-11, University of Oregon, 2006.

J. Li, S. Stafford, and T. Ehrenkranz, "SWORD: Self-propagating Worm Observation and Rapid Detection," Technical Report CIS-TR-2006-03, University of Oregon, 2006.

R. Rejaie and S. Stafford, "A Framework for Architecting Peer-to-Peer Receiver-driven Overlays," in *Proceedings of the International Workshop on Network and Operating System Support for Digital Audio and Video*, pp. 42-47, 2004.

ACKNOWLEDGEMENTS

I am grateful for the support and patience I received from Rebecca, Sophie, Lillian, and the rest of my family.

I would also like to thank my advisor, Jun Li, for all of his help and encouragement.

For my children. Stick with it.

TABLE OF CONTENTS

Chapter Page

I. INTRODUCTION 1

1.1. Assumptions 2
1.2. Challenges 3
1.3. Key Contributions of This Research 4
1.4. Roadmap of This Dissertation 4

II. THE FUNDAMENTALS OF WORMS 6

2.1. Worm Scanning Mechanisms 6
2.2. Worm Infection Mechanisms 8
2.3. Case Studies 9
2.4. Defenses 13
2.5. Worm Damage Mitigation 14
2.6. Advanced Worms 16

III. A SURVEY OF EXISTING WORM DETECTORS 18

3.1. Desirable Features in a Detection System 18
3.2. Worm Detection Techniques 20
3.3. Detection Systems 54

Chapter Page

3.4. Detector Selection 61

IV. EVALUATION FRAMEWORK FOR WORM DETECTORS 66

4.1. Trace Conversion 66
4.2. Environment Generation 71
4.3. Worm Simulation 72
4.4. Evaluation 75
4.5. Experiment Coordination and Results Processing 76

V. A COMPARISON OF BEHAVIOR-BASED WORM DETECTORS . . 77

5.1. The Selected Worm Detectors 77
5.2. Detector Performance Metrics 84
5.3. Experiment Design 89
5.4. Results 96

VI. EVASIVE WORMS 112

6.1. Evasive Worm Capabilities 113
6.2. Evasion Tactics 114
6.3. Methodology 118
6.4. Metrics 119
6.5. Evasive Worm Detection Results 121

Chapter Page

VII. A NEW WORM DETECTOR 156

7.1. Preventing Fast Scanning 157
7.2. Quiescent Periods 159
7.3. Clustering 162
7.4. SWORD2 163

VIII. AN EVALUATION OF THE SWORD2 DETECTOR 164

8.1. Burst Duration Detector 165
8.2. Quiescent Period Detector 169
8.3. SWORD2 Detector Compared with Existing Works 173
8.4. SWORD2 vs Evasive Worms 177
8.5. Analysis 183

IX. CONCLUSION 185

9.1. Contributions 186
9.2. Future Work 188
9.3. Conclusion 191

REFERENCES CITED 192

LIST OF FIGURES

Figure Page

2.1. A stack buffer-overflow attack 8

3.1. Gateway deployment 20

3.2. Detection taxonomy 21

3.3. Double honeypot system 29

3.4. Static signature 31

3.5. Autograph architecture 32

3.6. PolyGraph's advanced signature 36

4.1. Trace conversion architecture 67

4.2. Connection extraction process 71

5.1. False positives against legitimate traffic 97

5.2. F- against random worm 101

5.3. Latency against random worm 102

5.4. F- against local-preference worm 103

5.5. Latency against local-preference worm 106

5.6. F- against topo worm 108

5.7. Latency against topo worm 109

6.1. Effective scanning rate vs DSC 124

6.2. Evasion rate vs DSC 125

6.3. Maximum effective rate vs DSC 126

6.4. Effective scanning rate vs MRW 128

6.5. Evasion rate vs MRW 129

6.6. Maximum effective rate vs MRW 130

Figure Page

6.7. Effective scanning rate vs RBS 132
6.8. Evasion rate vs RBS 133
6.9. Maximum effective rate vs RBS 134
6.10. Effective scanning rate vs PGD 136
6.11. Evasion rate vs PGD 137
6.12. Maximum effective rate vs PGD 139
6.13. Effective scanning rate vs TRW with 100 known targets 142
6.14. Evasion rate vs TRW with 100 known targets 143
6.15. Effective scanning rate vs TRW with 1000 known targets 144
6.16. Evasion rate vs TRW with 1000 known targets 145
6.17. Maximum effective rate vs TRW 147
6.18. Effective scanning rate vs TRWRBS with 100 known targets 150
6.19. Evasion rate vs TRWRBS with 100 known targets 151
6.20. Effective scanning rate vs TRWRBS with 1000 known targets 152
6.21. Evasion rate vs TRWRBS with 1000 known targets 153
6.22. Maximum effective rate vs TRWRBS 154
7.1. Examples of observed connections over time 161
8.1. F- and detection latency for BDD with no clustering 167
8.2. F- and detection latency for BDD with clustering 169
8.3. F- and detection latency for QPD with no clustering 172
8.4. F- and detection latency for QPD with clustering 173
8.5. F- and detection latency for SWORD2 175
8.6. False Negatives for all detectors 176
8.7. Effective scanning rate vs SWORD2 179
8.8. Evasion rate vs SWORD2 180

Figure Page

8.9. Maximum effective rate in enterprise environment 181

8.10. Maximum effective rate in campus environment 182

8.11. Maximum effective rate in department environment 182

8.12. Maximum effective rate in wireless environment 183

LIST OF TABLES

Table Page

2.1. Worm case studies 10

3.1. Desirable worm detector attributes with measurable metrics 19

3.2. Worm detection techniques (in order of publication year) 55

3.3. Accuracy and speed of selected systems 56

3.4. Detection technique coverage 59

3.5. Worm detector capabilities 60

3.6. Pros and cons of categories of worm detection 62

4.1. Flow definition 69

4.2. Data elements captured about each flow 70

5.1. Metrics 87

5.2. Trace statistics 93

5.3. Parameter choices for the detectors 100

6.1. Evasive worm capabilities 115

8.1. Parameter choices for the Burst Duration Detector 166

8.2. Parameter choices for QPD 171

8.3. Parameter choices for SWORD2 174

8.4. Average detection latency for all detectors 177

LIST OF CODE LISTINGS

Code Listing Page

6.1. DSC Blind Evasive Worm 123

6.2. DSC Perceptive Evasive Worm 123

6.3. MRW Blind Evasive Worm 126

6.4. MRW Perceptive Evasive Worm 127

6.5. RBS Blind Evasive Worm 131

6.6. RBS Perceptive Evasive Worm 131

6.7. PGD Blind Evasive Worm 135

6.8. PGD Perceptive Evasive Worm 135

6.9. TRW Blind Evasive Worm 141

6.10. TRW Perceptive Evasive Worm 146

6.11. TRWRBS Blind Evasive Worm 148

6.12. TRWRBS Perceptive Evasive Worm 155

CHAPTER I

INTRODUCTION

The Internet plays a critical role in the operation of business, government, and even our personal lives. Because of its importance, we must ensure that it is reliable and available. Network worms are a form of *malware* that can compromise the integrity and availability of the Internet. This dissertation focuses on detecting worms, a critical component in defending the Internet against the damage they can cause. Extensive research has been done in this field, but recent worms such as IKEE.B and StuxNet have raised questions about our ability to detect modern, evasive worms. This dissertation seeks to answer those questions, and introduces a new worm detector that provides superior performance against evasive worms.

It is almost impossible to overstate the importance of the Internet in our daily lives. It is almost equally difficult to overstate the number of new devices that rely on the network that are activating every day Andy Rubin, of Google, reported in December of 2011 that over $700,000$ new Android devices were being activated every day[1]. Over the Christmas weekend in 2011, he reported that over 3.7 *million* devices were activated [2]. This is a phenomenal number of new devices, and all of these devices need to use the Internet to operate effectively.

Advanced mobile operating systems, such as Apple's iOS include sophisticated measures to prevent malicious use of the hardware. However, these protections can be circumvented by users wishing to run other software. In early 2012, computer news website Gizmodo.com reported that during the first three days a new "jailbreak" of Apple's successful iPhone 4S and iPad 2 was available, nearly one-million people used the jailbreak software [3]. Installing unsupported operating

systems such as this was the sole reason for the outbreak of the IKEE.B worm. There is nothing the software vendors can do to protect user's from this sort of risk.

A worm that utilized these jailbroken machines could cause significant disruption to the Internet. These facts highlight the importance of effective worm defenses. Specifically, we must be able to identify worm infected hosts without needing to run software directly on each host machine. We must not rely on specific byte-stream signatures to detect worm traffic because the traffic may be encrypted, as was the case with IKEE.B. And we must be adaptive to new applications and protocols, because users are downloading and installing more software then ever before, iOS users have an App store with hundreds of thousands of applications available for download [4]. To fight the threat that network worms pose, we need to advance the state of the art of worm detection.

1.1. Assumptions

We make the following assumptions in this work:

We assume that despite advances in defensive measures such as address space randomization and protected memory, there will always be vulnerabilities in networked computing systems. This assumption seems to be a safe one based on the decision making typical of end-users. The iPhone worm and "jailbreaking" numbers referenced above are an an example of users unwittingly installing insecure software on a network attached device. It is hard to imagine them giving up this behavior.

In addition, we assume that IPv4 based networking will remain the standard in the near future. The IPv4 standard will inevitably be replaced, and depending on the nature of the replacement, the characteristics of network worms may

change greatly. For example, if Content Centric Networking [5] becomes the norm, scanning for vulnerable hosts changes dramatically. A move to IPv6 will also change host-level behavior, but it is unclear how this will play out in practice. However, it is clear that any change in the core networking standards will take time to complete, so the current worm threat will remain for the near future.

Finally, we assume that network operators are interested in detecting malicious behavior on their network. This is perhaps our biggest and riskiest assumption. It is unclear whether worm detection systems are broadly deployed, and if they are not deployed, whether it is due to their poor performance or some other reason. It seems logical to us that a network operator would want to be aware of malicious activity on their network and so would deploy software to detect it, but there is no practical way to verify this.

1.2. Challenges

There are a number of challenges inherent in worm detection research.

The first challenge is to define a scope and set of scenarios that enable us to make meaningful evaluations. The field of worm detection is a broad one, and there is no way to cover every aspect of it in detail. We have seen that worms can be a threat to the Internet as a whole, but they also threaten individual organizations. A worm can carry a malicious payload that can steal or delete important data. The first step in defending against worms is knowing that they are present in a network. Towards that end, in this work we have chosen to focus on detecting worms in small protected networks. This sort of detection is broadly applicable, unlike systems such as network telescopes like [6, 7] which require large address spaces to deploy). A large organization such as a university could use a detector

targeted at small networks by deploying an instance of it for individual subnets within their network. The opposite is not true however, a small network operator simply cannot make use of a detector that requires a large network to operate.

One of the most significant challenges for this work is finding suitable network traces. We would like to evaluate detectors against a broad range of traffic types, but due to privacy restrictions it is very difficult to acquire such traces. In this study we were forced to use some traces that are as much as 10 years old, which may not provide an accurate representation of modern traffic.

1.3. Key Contributions of This Research

In this dissertation we present the following contributions to advancing the state-of-the-art in worm detection research: we develop a framework to easily evaluate behavior-based worm detection systems, compare the performance of existing behavior-based worm detection systems to determine which performs the best, extend the evaluation to include evasive worms that deliberately try to avoid detection, and to use the principles learned in this evaluation to build a detector that outperforms existing worm detection systems.

1.4. Roadmap of This Dissertation

This dissertation is organized as follows. In Chapter II we discuss the fundamentals of network worms, outlining their lifecycle and presenting several case studies. Chapter III describes the desirable features of a worm detector, surveys existing worm detection systems, and discusses the evaluation parameters of worm detectors. We present our framework for evaluating worm detection systems in Chapter IV. We use this framework to evaluate six behavior-based worm detectors

in Chapter V, which was co-authored with Dr. Jun Li and published in the 2010 *Proceedings of the Recent Advances in Intrusion Detection (RAID) Symposium.* Chapter VI extends this evaluation by considering worms that attempt to evaded detection. Taking the lessons learned from evaluating existing worm detectors, we present the design of the SWORD2 detector in Chapter VII and evaluate its performance in Chapter VIII. Finally, we discuss future work and open issues along with our conclusions in Chapter IX.

CHAPTER II

THE FUNDAMENTALS OF WORMS

A computer worm is a self-propagating program. A worm running on a host will actively scan the network (or the entire Internet) that the host is connected to, looking for additional victims to infect. A worm infects a remote host by gaining sufficient privileges to copy itself to, and then execute itself on, the remote host. Privileges are typically gained by exploiting a flaw in software running on the remote host, but may also be acquired through poor configuration of software or services.

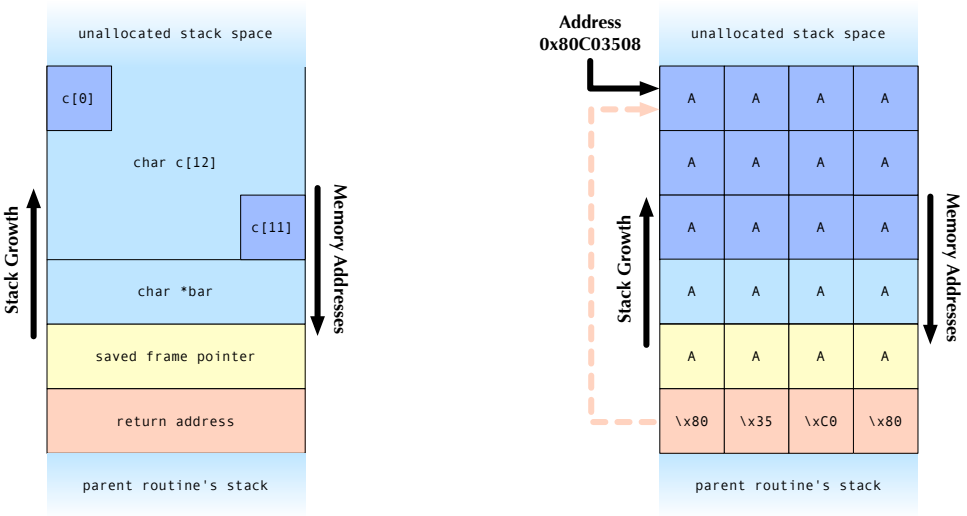
The activity of a worm can be broken down two major steps: (1) finding new hosts to infect, and (2) infecting newly found hosts. Finding hosts can be done in a variety of ways, but each host must eventually be contacted over the network. The infection step involves gaining privileges on the remote host, copying the worm code to it, and executing the code on it. In nearly all existing worms, all of these elements are combined into the single action of attempting malicious connections to remote addresses with the goal of infecting them. The SQL-Slammer worm is an excellent example of this: the UDP packets it sends are sufficient to compromise a machine, so combine the finding and infecting into a single action. Because UDP is a stateless protocol the source host does not need to establish a connection or wait for a response from a remote address before moving on to the next target, allowing SQL-Slammer to scan and infect at an astonishing rate.

2.1. Worm Scanning Mechanisms

Depending on the strategy for locating vulnerable hosts, the chance of

each connection finding a vulnerable host and infecting it may be low, as in the case of a worm that scans randomly, or high, in the case of a worm that has a hit-list of vulnerable targets. In their seminal work "How to 0wn the Internet in Your Spare Time" [8], Staniford, Paxson, and Weaver catalogued a variety of scanning mechanisms, including Random, Local Preference, Sequential, Permutation, Topological, and Hit-list scanning. The most basic mechanism is random scanning. Using this technique, worms simply generate a random 32-bit address (assuming standard IPv4 network addressing), and then try to connect to it. Random-scanning is simple to implement and effective, but is inefficient because many connections will be made to addresses where no host is present. A more efficient means of scanning, employed be several worms including Code Red II, is to focus efforts on addresses near the currently infected host. This leverages the fact that hosts are not evenly dispersed within the Internet, and that vulnerable hosts tend to be clustered [9]. Staniford et al. outlined additional scanning strategies that a worm can employ, including hit-list scanning, permutation scanning, and topological scanning [8]. A worm using hit-list scanning works from a pre-compiled list of vulnerable hosts, making it extremely fast. Permutation scanning allows hosts to compartmentalize their efforts, all infected hosts share a permutation of the address space, and have some section of that permutation assigned to them to scan. Finally topological scanning worms acquire their targets from information available from the infected host or on the network. This might include lists of recently contacted hosts, connections into some peer-to-peer network, or even search results from a major search engine [10].

The scanning mechanism impacts not only the speed with which a worm propagates, but also may change the way it appears on the network. For example,



(a) Typical allocation of the stack

(b) The stack with buffer overflowed

FIGURE 2.1. **A stack buffer-overflow attack** on buffer c writes data beyond what is allocated, overwriting the stack pointer and return address. A real attack would write its payload shell-code to in place of the A characters.

a random scanning worm will typically generate many failed connection attempts, while a topological worm will produce a much smaller number.

2.2. Worm Infection Mechanisms

There are many possible avenues that would allow a worm to take over a remote host, but by far the most common is exploiting a buffer overflow. Despite this style of attack having been a known vector for worm propagation for many years, buffer overflow vulnerabilities continue to show up frequently in software. It does not appear that they will be eradicated any time soon.

The essence of a buffer overflow attack is to write more data to a buffer than it has allocated space for. The excess data will then overwrite adjacent memory addresses, and when this is done properly, the overwritten memory areas can be used to execute arbitrary code. Buffer overflow attacks must be targeted specifically

at an architecture and operating system. The buffers that are overwritten can be either on the heap or the stack, with different exploitation requirements for the two options. The heap is the pool of free memory that is allocated dynamically to the running program. It is typically referenced indirectly. The call stack stores the information about the execution of the program, but varies greatly with operating system and machine environment. One typical function of the stack is to store the address to which each function should return control when it finishes executing. See Figure 2.1 for an example of using a buffer overflow to overwrite the return address of a stack frame.

A second form of attack is known as code injection. It is typically found in web applications. In this attack, a server accepts posted data from a client, and if it doesn't properly sanitize the data for code markers, it can end up executing the posted data as code. This allows the client the opportunity to execute arbitrary code on the server, allowing the client to compromise it and infect it with a worm.

2.3. Case Studies

To get a better sense of the threat, let us examine a few of the more well-known worms that have been discovered in the wild. Table 2.1 summarizes the details.

- *Code-Red v2 (2001):* Released on July 19, 2001, this worm infected approximately 359,000 hosts in just 14 hours [11], with damages estimated at more than two billion dollars [16]. It exploited a *buffer-overflow* vulnerability in Microsoft's IIS Web Server to inject code which defaced web pages, randomly scanned the Internet for additional hosts to infect, and periodically issued a denial of service attack against `www1.whitehouse.gov`.

TABLE 2.1. **Worm case studies**

Worm	**Scan Type**	**Vector**	**Payload**	**Victims**
Code Red v2	Random	Buffer	Defacement, DDOS	$\sim 359K$ [11]
Code Red II	Local Pref.	Buffer	Install backdoor	$\sim 359K$ [11]
Sql Slammer	Random	Buffer	None	$\sim 75K$ [12]
Witty	Hitlist, Rand.	Buffer	Erased HD	$\sim 12K$ [13]
Santy	Topological	Injection	Defacement	$< 20K$ [10]
Conficker	Various [14]	Buffer	BotNet client	$> 4M$ [15]
IKEE.B	Random	Password	BotNet client	Unknown
Conficker	Various	Multiple	Industrial sabatoge	Unknown

– *Code-Red II (2001):* This worm exploited the same vulnerability as the original Code-Red worm, but was otherwise unrelated. It did not deface web pages or issue denial of service attacks, instead installing a back door allowing root access to the infected machine. The Code-Red II worm is notable because it used a more sophisticated scanning mechanism than previous worms. Rather than scanning the entire Internet randomly, it preferentially scanned local networks. Because vulnerable hosts tend to be clustered, this scanning method improved the propagation speed of the worm [17, 18, 19].

– *SQL-Slammer (2003):* One of the fastest spreading worms observed in the wild, the SQL-Slammer worm infected most of the vulnerable population of approximately 75,000 hosts (servers running the vulnerable version of the SQLServer software) in under 10 minutes. It propagated quickly because each scan required only a single, small UDP packet; allowing high-bandwidth victims to scan at enormous rates. The peak aggregate scanning rate was approximately 55 million scans per second, which was sufficient traffic to disrupt significant portions of the Internet [12]. It is remarkable that SQL-Slammer achieved this much damage despite a naive random scanning

algorithm and a small vulnerable population. It serves a reminder that even a small number of infected hosts can cause a great amount of damage.

- *Witty (2004):* The first widely propagated Internet worm to carry a destructive payload, the Witty Worm was notable in that it took advantage of a vulnerability in security software in order to propagate [13]. The Witty worm scanned for victims randomly, stopping after 20,000 scan packets to make detection harder. After scanning it began writing random data to the hard-disk of the infected machine.

- *Santy (2004):* The first widely propagated *search* worm, Santy found vulnerable hosts by submitting specially formulated search queries to the Google search engine (later variants used a variety of search engines) to find web servers running a vulnerable version of a PHP based bulletin board software. The worm defaced the websites that it infected, but carried no other malicious payload. The number of infected servers is estimated in the low thousands [10], but no hard figures have been published.

- *Conficker (2008):* The Conficker worm infects Microsoft Windows machines by crafting a malicious RPC (remote procedure call) request. It has successfully spread to millions of machines [15] and continues to evolve in response to defensive measures taken [20]. Conficker uses a variety of scanning techniques including topological, local preference, and random scanning. It has additional advanced features including: employing a variety of techniques to hide itself within infected systems, deleting system rollback points to prevent easy removal, actively disabling installed antivirus software, and dynamically updating itself.

– *IKEE.B, the iPhone Worm (2009):* The IKEE.B worm targets "jail-broken" iPhones [21]. These phones have had the default operating system replaced by their user with a new operating system downloaded from the Internet. "Jailbreaking" a phone in this way allows users to run software that would not otherwise run on their system. Unfortunately, the jailbreak software installed a default root password which was exploited by the worm. The worm installed bot-net software and scanned network prefixes know to be run by mobile phone operators who carry the iPhone. The worm itself was relatively simplistic, but because it spread via ssh (an encrypted channel) and exploited a configuration weakness rather than a buffer overflow, it was invisible to host-based and content-based detector systems.

– *StuxNet (2010):* The StuxNet worm is the most sophisticated worm seen to date, utilizing multiple zero-day vulnerabilities to attack industrial control systems [22]. It was likely targeted at a uranium processing plant in Iran. It utilizes a variety of mechanisms to propagate and attempts to cover its tracks when it does infect a system. It is difficult to quantify the damage caused by this worm, because it seems to be targeted a specific plant, but given the level of sophistication it shows, it could have caused tremendous damage if targeted at the Internet in general.

Early worms were exceedingly primitive and detecting them posed little difficulty. The sophistication of StuxNet shows that this will not always be the case. Worms like IKEE.B can propagate over encrypted channels and don't rely on buffer overflow attacks. Worms like StuxNet and Conficker attempt to hide all traces of their existence on infected hosts. Worms are advancing in sophistication and worm detection methodologies must accordingly keep up.

2.4. Defenses

Detection is not the only weapon in the arsenal against worms. There does exist technology that could somewhat mitigate the risk from worms, by eliminating some avenues for a compromising a host. These will not entirely eliminate the risk from worms as other avenues will always remain open. The dominant majority of the worm attacks in the wild exploit buffer overflows, but compiler technology can ensure that many such vulnerabilities are eliminated from software.

StackGuard [23] is a compiler modification that protects programs against buffer overflow in the majority of cases with only a small performance penalty. A more advanced version is CCured [24], which protects all memory accesses, but these technologies must be adopted into widespread usage before they will help.

Another possibility is that of address space layout randomization. In this scheme, the locations of various libraries and system calls in memory is randomized, making it more difficult for an attacker to call specific functions from buffer overflow attacks. Both Microsoft Windows Vista and Mac OS X.5 implement a form of this, in theory making it harder to take advantage of buffer overflows. Research by Schachem et al. showed that this may only slow attackers, and not stop them entirely [25].

The preceding tools limit the ability of a worm to *infect* hosts, but work by Anatos et al. [26] show that is is also possible to reduce the ability of worms to *find* hosts to infect. Instead of randomizing address locations, they suggest randomizing the assigned addresses of servers on a regular basis to defeat the ability of worms to have a *hit-list* of vulnerable servers to attack.

Once a worm has discovered vulnerable hosts and exploited their vulnerability there are additional steps that can be taken. Infected hosts can also be quarantined

to limit their ability to infect additional targets [27, 28, 29]. Infected hosts can be restored and patched to prevent reinfection. Brumley et al. examined the tradeoffs between the above mentioned protection mechanisms vs patching and blacklisting hosts [30]. They showed that the most useful approach is a hybrid of reactive antibodies (which includes signature based defenses as well as active system patching) coupled with protective measures such as address space randomization. They showed address blacklisting requires unrealistically fast reaction times and that local containment required an extremely high deployment rate to be effective.

Unfortunately all of these solutions provide only limited mitigation from worms and broad adoption of them will take time. The threat from worms is here for the foreseeable future.

2.5. Worm Damage Mitigation

Detecting the presence of a worm is not only task that must be accomplished. Once we know that a worm is present, we must do something about it. This section discusses the research that has been done into mitigating the damage caused by worms. The solutions presented here generally take one of a couple of forms, some suggest *throttling* the worm to reduce the speed at which it propagates, others attempt to block only worm connections from an infected host, while finally some solutions suggest blocking all connections from an infected host.

One of the earliest mitigation works [31], titled LaBrea after the LaBrea Tar Pits, is a honeypot which responds more and more slowly to inquiries, keeping the remote host occupied so it can't propagate more widely. It was followed by Williamson's introduction of the notion of throttling connection rates (limiting a hosts ability to connect to many destinations quickly) to slow the spread of

worms [32]. This work was further explored by Twycross and Williamson in [33], and then extended to email viruses in [34].

Worm detection works that rely on observing connection failures can be easily extended to reduce the rate of worm spread by employing credit base rate-limiting. Schecter et al. [35] employ this method with their reverse sequential hypothesis testing work. Each host is issued a starting balance of 10 credits, each connection they open to a host not in their recent set of destinations costs one credit. That credit is returned (plus an additional one) if the connection is successful, but is not returned if the connection fails. Additionally, hosts receive a small allowance of credits periodically if they exhaust their supply, and they are forced to surrender some of their surplus credits if they acquire more than 15. In this way, a host is prevented from making large numbers of connections that fail, while still allowing for occasional normal failures. A similar solution based on Threshold Random Walk was implemented by Weaver et al. [27].

Wong et al. studied the deployment effectiveness of throttling at backbone vs edge routers [36]. They showed that it is dramatically more effective to deploy rate limiting at the backbone than at edge routers.

A different approach is taken by Gu et al. in [37], which actively blocks connections from hosts after they are discovered to be infected. They showed that even blocking traffic entirely from infected hosts will not stop the spread of the worm, though it will slow it down considerably. Kannan et al. showed that if firewalls cooperate at blacklisting hosts, they can be much more effective [38], even with as few as 10% of the firewalls participating.

Moore et al. examined the problem in a general sense, looking at the reaction time, containment strategy, and deployment scenario [39]. They showed that a

fast reaction time is vital, that content filtering is more effective than address blacklisting, and that a significant portion of the largest Autonomous Systems (such as Internet service providers) must cooperate to effectively block the spread of worms. Ganesh et al. used game theory to examine the best strategy for worms and detectors [28]. They showed that a Bayesian approach to combining information from multiple detectors can be effective, while for a worm, slow scanning is the most effective strategy.

In addition to quarantine, infected hosts can be restored and patched to prevent reinfection. Brumley et al. examined the tradeoffs between the above mentioned protection mechanisms vs patching and blacklisting hosts [30]. They showed that the most useful approach is a hybrid of reactive antibodies (which includes signature based defenses as well as active system patching) coupled with protective measures such as address space randomization. They showed address blacklisting requires unrealistically fast reaction times and that local containment required an extremely high deployment rate to be effective.

2.6. Advanced Worms

The threat from worms is substantial, and is most likely only going to get worse as the speed of propagation and the stealthiness of worms increases along with their countermeasure abilities against worm detection mechanisms. In cataloging potential worm scanning techniques, Staniford et al. outlined the great speed with which worms can infect the Internet and the vast threat that they pose [8]. This analysis was taken even further by Weaver and Paxson in 2004, estimating that a well engineered worm could cause upwards of $50 Billion dollars in damage and lost productivity [40]. Such a worm could move frighteningly

quickly, potentially infecting 95% of Instant Messenger clients in as little as 510 milliseconds [41].

There are many ways for a worm to be more stealthy in its spread as well. Polymorphic worms change the code that they send across the network, limiting the effectiveness of signature-based systems at detecting them. Song et al. showed that polymorphic shellcode used in buffer overflow attacks can be disguised with off the shelf polymorphism engines to appear as essentially random bytes [42]. Van Gundy, Balzarotti, and Vigna furthered this attack, showing that web-based attacks based on PHP exploits can also be extremely polymorphic [43]. These random bytes can be manipulated to show a normal looking byte frequency distribution by a blending attack, and described by Fogla et al. [44]. Multi-attack vector worms that are capable of exploiting more than one vulnerability add an additional layer of complexity, as detectors can't focus on a single attack. Worms can also use intelligent scanning to avoid some types of detection mechanisms all together, as in the research by Rajab, Monrose, and Terzis [45].

Many of the detection mechanisms we will discuss rely on training against legitimate traffic. This makes them susceptible to malicious training attacks that can cause them to treat legitimate traffic as worm traffic. Attacks of this nature have been described against several detection mechanisms ([46, 47]), and has been formally defined as a "learning problem in an adversarial environment" by Newsome, Karp, and Song [48]. These works show that as worms get smarter, detecting them gets more difficult and dangerous.

CHAPTER III

A SURVEY OF EXISTING WORM DETECTORS

3.1. Desirable Features in a Detection System

The threat from worms is significant, but we are not defenseless in this battle. With a basic understanding of worms, their life-cycle, and their capabilities, we can now turn our attention to methods of detecting the presence of worms within a network. In this section, we describe the desirable attributes of worm detection.

There are many desirable characteristics in a worm detection system (see Table 3.1 for a summary). These attributes provide us with useful metrics to consider as we examine proposed worm detection systems. The most obvious desirable feature in a worm detection system is that it must be accurate. This means that it must have a low *false negative* rate, meaning that it rarely fails to detect worm activity; as well as a low *false positive* rate, meaning that it rarely identifies legitimate activity as a worm.

Accuracy alone does not sufficient define a system as good or bad. It must also offer *comprehensive coverage*, which is to say that it should detect all of worms regardless of scanning type, infection vector, network protocol, or detection countermeasures the worm may employ. A worm detection system should be fast, meaning that it is able to detect a worm with a minimum of worm activity.

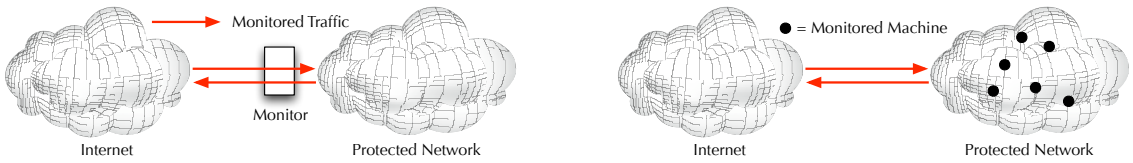
Worm detectors should also be easily deployable. A single deployment location that can protect an entire network — such as at a network gateway — is more desirable than requiring deployment on each machine in a network (see Figure 3.1) as it would require less overhead to install and maintain. Additionally, the system must not impose onerous overhead. If deployed at a network gateway

TABLE 3.1. **Desirable worm detector attributes with measurable metrics**

Desired Attribute	**Measurable Metrics**
Accuracy	False negatives (worm activity identified as legitimate) False positives (legitimate activity identified as worm)
Coverage	Scan types detected Infection vectors detected Network protocol limitations
Speed	Detection latency
Deployability	Deployment location Runtime overhead
Capabilities	Infected hosts identified Worm connections identified (worm signature) Vulnerability exploited identified

this means that it must be able to keep up with the volume of traffic on the network. If deployed on an individual machine it must not overly impact the performance of the software running on the machine.

Finally, the more information a worm detector gains, the more useful it is. The minimum amount of meaningful information a detector can supply is simply that a worm is present somewhere. This information is somewhat useful in that it would raise the alert level of network administrators, but does not provide any detail for them to act on. On the other hand, the maximal amount of information a detector could provide would be to identify each individual network packet as worm or non-worm, and to provide detailed information about the exploit the worm is taking advantage of. A system generating this much information would certainly give network operators information to combat the worm. A detector generating that much useful data could potentially take automatic action itself against the worm, perhaps blocking worm connections or even patching vulnerable hosts.



(a) Gateway Deployment (b) Host-level Deployment

FIGURE 3.1. **Gateway deployment** is more desirable for a worm detection system because it requires less overhead for installation and maintenance.

3.2. Worm Detection Techniques

We now examine each worm detection technique individually, as they are applied in various detection systems. After describing each technique, we briefly analyze its strengths and weaknesses towards worm detection.

There have been a variety of worm detection system proposed, using a wide range of techniques. We make the distinction here between a detection *system*, a relatively complete structure for detecting a worm which is typically the subject of one or more research publications; and a detection *technique*, which is a specific, low-level means of detecting one aspect of a worm. Worm detection systems typically employ multiple techniques. Looking directly at the techniques allows us to consider their strengths and weaknesses beyond the constraints of the system they are implemented in.

To examine worm detection techniques, we first broadly categorize the detection techniques (Figure 3.2) into one of four categories: host-based, honeypot-based, content-based, or behavior-based. In Section 3.2 we summarize each technique in the context of the published systems which used it; then in Section 3.3 we examine the performance of each technique, again in the context of the published detection system.

The four categories of worm detection techniques are as follows:

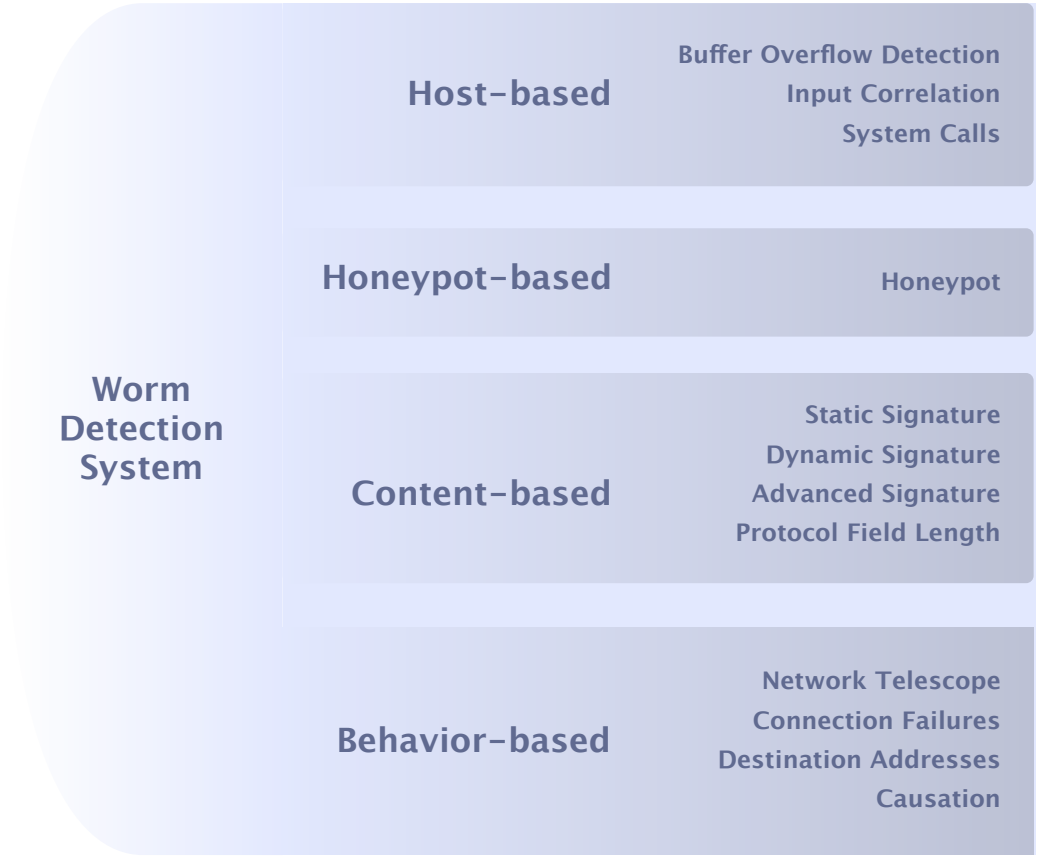


FIGURE 3.2. **Detection Taxonomy**

– *Host-based:* Host-based detection is characterized by the fact that it uses information only available at the end-host. It must be installed on each host that is to be protected by it. Modifications may be required to the operating system or the software that runs on it to give the detection software access to the internals of the execution environment. Host-based techniques include: buffer overflow detection, correlating network data to memory errors, and looking for patterns in system calls.

– *Honeypot-based:* Honeypot-based systems place a vulnerable host on the network that provides no real services. Because it provides no services, any traffic to the honeypot can immediately be considered suspicious.

– *Content-based:* Content-based systems observe the contents of network traffic looking for byte patterns that match the signature of a worm. The signatures are either generated on the fly by the worm detector, or developed manually from deconstruction of a worm instance. They rely on the fact that some aspect of the data that is sent to take advantage of a vulnerability is never sent as part of legitimate traffic, and can therefore be used to accurately identify worm traffic. Content-based techniques include static signatures, dynamic signatures, and advanced signatures.

– *Behavior-based:* Behavior-based systems work by observing the network behavior of end hosts and identifying patterns that are indicative of the presence of a worm. Behavior-based techniques include: connection failures, network telescopes, pattern of destination addresses, and causation.

Host-level Techniques

Host-based worm detection is based on modifying the the software running on a particular host to allow the observation of its internal state. Software modified in such a manner is referred to as being "instrumented", indicating that monitoring instruments can observe its actions, which in turn allows the detection software to spot the activity of the worm as it interacts with the operating system or deployed software.

All host-based systems suffer somewhat in deployability, because they require deployment on each host that is to be protected. If a host-based system is employed only on the web servers within an organization, and a worm is launched that attacks only mail servers, that organization might never detect the presence of the worm. On the flip side, host-based solutions generally have high accuracy

because they have access to all the inner workings of the host and can know with high accuracy when they are infected.

Buffer Overflow Detection

Most existing worms exploit buffer overflow vulnerabilities to propagate. One way to detect a worm is simply to watch for these buffer overflows to occur. Buffer overflows can happen during normal operations, however, so there must be some way to limit false positives. The COVERS [49], Sweeper [50], and HoneyStat [51] all use buffer overflow detection from a system such as StackGuard [23] to trigger their worm detection.

Assessment: Systems relying on buffer overflow detection for worm defense have two significant limitations. They do not have comprehensive coverage, being unable to detect worms that exploit a non-buffer-overflow vulnerability A worm like the Santy worm, which uses code injection to gain access to remote machines, would go undetected by a system relying on buffer overflows. Additionally, unless coupled with other techniques, buffer overflow detection gives virtually no information about the nature of the attack. It can identify that a host has been compromised, but give no more information than that.

Correlating Network Data to Memory Errors

Simple buffer overflow detection by itself doesn't yield much insight into the what caused the actual problem. Because of this, it is generally coupled with some other technique to further refine the findings. One such method is to in some way correlate network data to the memory error such that the source data that caused the buffer overflow can be employed in signature generation.

The COVERS system by Liang and Sekar [49], attempts to determine the root cause of a buffer overflow by forensic analysis of the system memory. COVERS consists of four phases: attack detection, correlation to input, identifying the input context, and signature generation. Attack detection works via a buffer overflow monitor such as StackGuard. Once an attack is detected, a forensic analysis of the host's memory is initiated. This is based on the observation that memory error attacks typically involve pointer corruption, and must include the corrupted pointer value in the data. COVERS attempts to correlate the data used in the buffer overflow attack with the network traffic received by that host. Once the attack data is known, the input context is narrowed by employing protocol parsers, helping to limit false positives. Finally a signature can be generated and disseminated. Signatures generated by COVERS rely on knowing protocol information, and specify the length of input fields and the byte-value distribution contained by them.

The TaintCheck system introduced in 2005 by Newsome and Song [52] labels data from untrusted sources as "tainted", then tracks the propagation of that data in memory and monitors whether or not it is used in dangerous ways. This has the advantage of not requiring specially compile binaries or the source code to applications to be protected. TaintCheck works by running programs in an emulation environment that allows all operations of the program to be monitored. This allows data to be monitored and tracked through memory, and to raise an alert before tainted data is used in ways deemed dangerous. Examples of dangerous activities include using tainted data as a jump target, as a string format argument, or as a system call argument. Once a dangerous activity is detected, TaintCheck can log the questionable data and the path it took through the system. This data could then be redirected to a signature generator.

Another system introduced in 2005, DACODA [53], works in a very similar fashion. DACODA uses full-system symbolic execution to track data from the network and observe malicious memory manipulations. Every byte of data read from the network is labeled with a unique label enabling DACODA to determine whether this data is used in a conditional flow transfer. If an oracle (such as Minos [54]) determines that the flow transfer is illegal, DACODA can export the data to a signature generator.

Vigilante [55] is a third contemporaneous system that instruments memory to detect worm attacks. It also introduces the notion of self-certifying alerts to allow hosts that detect worm attacks to broadcast that information to other hosts.

Tucek et al. combine several existing techniques in their Sweeper [50] work. They recognize that systems like TaintCheck, DACODA, and Vigilante impose a significant runtime overhead, making them problematic to deploy to live servers. To combat this, Sweeper employs a lightweight monitoring during normal usage, then reverts to more heavyweight analysis when an attack is detected. This utilizes the best attributes of both techniques and avoiding the onerous runtime overhead that plagues heavyweight analysis systems. They rely on address space randomization (by using a system such as StackGuard) for their light-weight attack detection, coupled with checkpointing and rollback to resume operation when an attack is encountered. The attack is then analyzed with a variety of tools including static core dump analysis, memory bug detection, dynamic taint checking [52], and dynamic backward slicing. The analysis then enables the production of signatures and execution filters.

Assessment: This technique is again very effective at against attacks that actually use memory errors to gain the ability to execute code on a host. It suffers

from the same lack of comprehensive coverage that plagues buffer overflow based detection. Worms that exploit some vulnerability that is not monitored by the detection system will escape detection entirely. In addition, the implementation of this technique is typically computationally expensive. TaintCheck, DACODA, and Vigilante all require executing applications in some form of emulator causing both runtime overhead as well as stability concerns. Because runtime performance and stability are key attributes desired by operators, these systems may not be deployed widely. Sweeper proposes a low-overhead solution by employing light-weight buffer overflow detection with a rollback/re-execute strategy for doing in depth analysis after an attack is detected, but even this imposes up to a 5% performance hit, and an unknown impact to stability.

System Calls

This technique detects worms by examining the system calls that are made by programs as they execute. A work by Malan and Smith [56] compares the pattern of system calls on one machine with the patterns exhibited by its peers, determining that a worm is present when the patterns overlap substantially. They exploit the fact that a given worm will make a series of system calls during execution that is relatively static across both time and instances of the worm, and that normal host operation does not exhibit this consistency. In order to determine if a host is infected with a worm, it exchanges snapshots of its system call activity with its peers. It then determines the similarity of its own snapshot with that of its peer's by treating each snapshot as an unordered set of system calls and calculating the percent of intersection between the snapshots. If more than 90% of snapshots show an intersection rate of greater than 50%, then both hosts are considered to be

infected with a worm. This method can incur false positives however, which must be limited by excluding legitimate applications from consideration. A followup work exploits the temporal consistency between worm instances to further improve accuracy [57].

Assessment: This technique has a serious limitation that is acknowledged by Malan and Smith in their work, it is prone to false positives [56]. They suggest that white-listing applications may be sufficient to reduce the false positives to manageable levels, but it is unclear how effective this would really be. When an application is white-listed, its system calls are no longer included in the pool that is compared with peers, but if that application is exploited by a worm, then the worm won't be detected. Any application that is white-listed is not protected, and if many applications must be white-listed the overall utility of the defense is reduced substantially. The system also generates the minimum possible amount of information about the attack, indicating that a worm is present, but no more than that. This system also requires that at least one other peer of a host be infected before detection can happen. This may significantly slow overall detection of the worm.

Honeypots

Honeypot-based worm detection is closely related to host-based detection, but differs in that host-based detection is deployed to live servers whereas honeypots by design serve no function beyond worm detection. All host-based worm detection methods could be deployed to the software running on a honeypot, but this is not generally necessary as all connections to a honeypot are already considered to be suspicious.

Honeycomb [58] was among the first honeypot systems to automatically generate signatures from traffic directed at it. It uses a simple longest common substring algorithm to spot similarities in packet payloads across multiple connections, and was successful at generating precise signatures for the CodeRed II and Slammer worms in the wild. Tang et al. determined that the assumption that all traffic to honeypots is malicious or a deliberate attack is a flawed one [59] but proposed two detection models to more accurately determine whether connection activity represents an actual attack.

The HoneyStat work by Dagon et al. [51] takes a slightly different approach. It uses a buffer overflow detector like StackGuard [23] to detect malicious activity, then couples this with disk and network monitoring. It generates a series of event notices which are forwarded to an analysis node that checks them and determines whether or not to issue a worm alerts and develop a signature. The events generated fall into the following categories: malicious memory events such as buffer overflows, unexpected network events such as outgoing traffic (the honeypot makes no outgoing connections of its own), and disk events such as critical system files being overwritten. The analysis node then employs a logit analysis to determine which events stem from a possible worm attack.

The use of honeypots was furthered in work by Tang and Chen [60]. They used a double-honeypot system like the one depicted in Figure 3.3. Incoming traffic to the network is directed to one honeypot. Outgoing traffic from that honeypot is directed to a second honeypot. This ensures that the second honeypot only sees traffic generated by the worm, helping to isolate that traffic and give a clean sample from which to build a signature.

Assessment: Perhaps the biggest limitation to honeypots is that despite

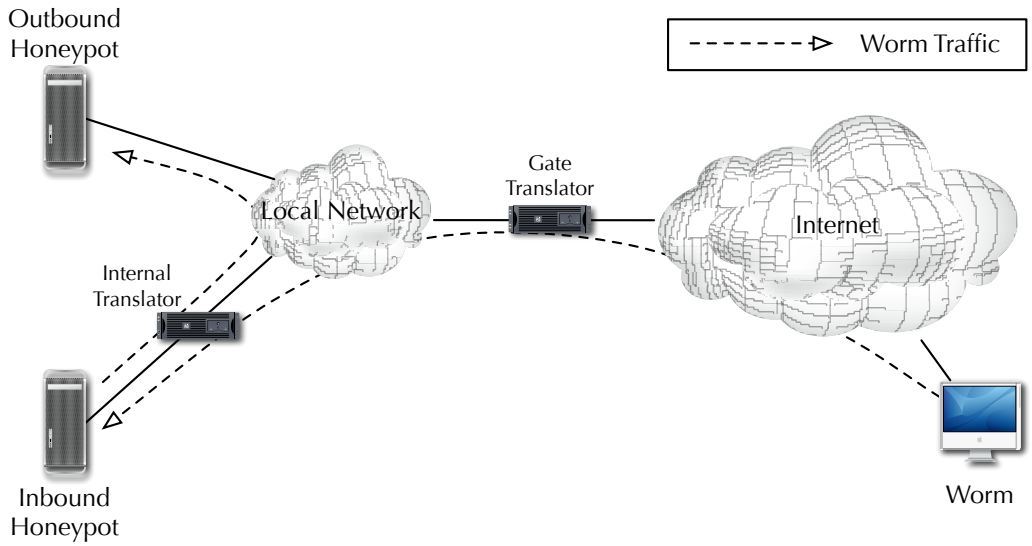


FIGURE 3.3. **The double honeypot system** directs incoming traffic to a honeypot that can be infected by worms. The outgoing traffic from this honeypot is directed to a second honeypot, and can be used to generate signatures. (Graphic adapted from [60])

serving no resources to legitimate users, they must attract connections from worms and be providing the vulnerable service that the worm takes advantage of. Non-targeted scanning worms can have their connections directed to honeypots relatively easily, as all unused network addresses can be redirected and delivered to a honeypot. The same is not true for more sophisticated worms like a topological worm. For a honeypot to detect the Santy worm, for example, it would need to be running the vulnerable software *and* be in a search engine's index.

Content-based Techniques

Content-based worm detection is based on the idea that when a worm exploits a given vulnerability, it typically does so by sending a carefully constructed message with some portion of *fixed content* that is unique to the exploit. Because this exploit-content is the same across each infection attempt and is not typically a

part of normal network traffic, it can be used as a signature to identify connections stemming from a given worm (see Figure 3.4 for an example of a static signature). Traffic containing such a signature is almost certainly generated by a worm, so once the signature is established, worm detection becomes straight-forward. Historically, signatures were established by researchers. They either observed the worm in action and manually identified the constant content, or they reverse engineered the worm's code to determine the signature. The following techniques all use this basic method, with more or less advanced ways of building and matching signatures.

Content-based systems possess many desired attributes. They can be situated at a network gateway, making deployment easy. They give a large information gain, identifying individual connections (and by extension the source hosts) as carrying worm infection. The biggest limitation on content based systems is that they do not provide coverage for polymorphic worms that change the appearance of their payload for each connection.

Static Signatures

There are a number of existing systems that employ static signatures in worm detection. Snort [61] and Bro [62] are two of the most popular and established open-source systems. They are designed to be deployed at a network gateway to monitor inbound and outbound network traffic. The Netbait [63] system extended this idea to use a collection of Snort-based detectors in a distributed fashion to provide relatively comprehensive information about which hosts on the Internet were infected with well known worms. Netbait is limited by its reliance on well known signatures: it cannot detect *zero-day* worms, those for which a signature has not yet been developed.

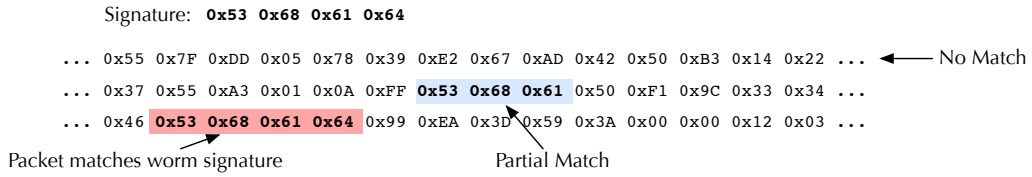


FIGURE 3.4. **A static signature** is a set of bytes (typically longer than the 4-bytes shown here) that can be watched for in network traffic. When a packet contains the full sequence of bytes (as in line 3) it matches the signature as is considered to be a worm packet.

Assessment: Static signatures will likely always have a place in worm defense due to their low deployment cost, but their utility is clearly limited by their inability to defend against zero-day worms.

Dynamically Generated Signatures

To overcome this reliance on pre-generated signatures, worm detection systems must be able to detect zero-day worms and automatically generate a signature for them. To do so, they first need to identify which network traffic is worm traffic, and secondly, use that traffic in developing a signature. Several systems use behavior-based techniques to identify suspicious traffic, then develop a signature based on the content in the suspicious flows.

WEW proposed a basic signature generation algorithm. It sends suspicious traffic (as identified by TCP scanning) to a honeypot and does longest common substring matching to produce a signature [64].

Autograph [65] and EarlyBird [66] are two contemporaneous landmark works in the field, using different methods to develop signatures dynamically based on analyzing the content of network traffic.

Autograph first identifies traffic as suspicious when that traffic stems from a a host that has attempted more than a set number of connections that fail.

The flow classifier is a pluggable component in Autograph, and it could in theory use a more sophisticated flow classifier to determine whether traffic is suspicious. Once the suspicious traffic is identified, it is grouped by target port with the traffic for each port being handled separately. Autograph then looks for the most frequently occurring byte sequences across the suspicious flows to a given port (see Figure 3.5 for an overview of the architecture). It is prohibitively expensive to search for prevalent byte sequences for any variable size sequence, so the flow contents are divided up into chunks using a content-based partitioning scheme that splits the flow on a predefined break-mark (such as a predefined byte-value). These data chunks are also bounded by minimum and maximum sizes to prevent short, non-specific signatures and long, overly specific signatures. Once the prevalent byte sequences are identified, they are filtered to include only those sequences that have originated from more than one source address. They are then evaluated against the suspicious flows until a set of signatures is identified that match against a configured portion of the population of suspicious flows. A blacklist is maintained of signatures that should not be allowed, in order to prevent signatures on common innocuous byte sequences that would then block legitimate flows.

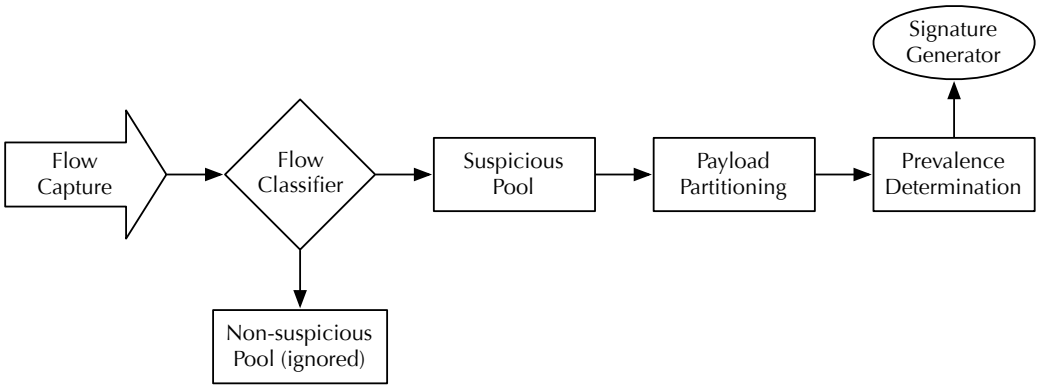


FIGURE 3.5. **Autograph Architecture**

EarlyBird also finds common byte sequences to generate a signature. However, rather than first identifying suspicious traffic and then limiting analysis to that, Earlybird analyzes all traffic that crosses a gateway. It identifies traffic as suspicious when it is prevalent (contains frequently appearing byte sequences) and highly dispersed (is associated with many distinct source and destination addresses). Suspicious byte sequences become a worm signature when they exceed given thresholds in both prevalence and dispersion. Real-time analysis of the volume of traffic crossing a fast gateway requires a number of optimizations. Like Autograph, EarlyBird cannot examine all arbitrary size byte sequences. It instead looks both at whole packets and all subsequences of the packet of a set size. Rather counting instances of the content itself, which would require large index tables, EarlyBird borrows techniques from the conceptually similar problem of finding high-bandwidth "heavy hitters". The heavy hitter algorithms track behavior based on a standard flow identifier (source/destination address, source/destination port, protocol). EarlyBird replaces this concept with the notion of a content identifier. These are 32-bit CRC values for the whole packet chunks and 40-byte Rabin fingerprints for the fixed size subsequences. These values are efficient to compute and require less storage space than using the full content itself. Sampling subsequences further reduces computational load. The prevalence counting serves as sort of a high-pass filter for EarlyBird, only once a chunk of content is shown to be prevalent is its dispersion tracked. This is done by recording the source and destination IP addresses involved with sending and receiving it. This recording happens in a memory efficient manner by hashing each content source and destination into a scaled bitmap. A scaled bitmap is a novel extension to existing bitmap techniques, where when the bitmap has too many bits set to retain

accuracy, an additional scaled bitmap is added that maps to a progressively smaller portion of the address space. These multiple bitmaps allow accurate counting to high numbers in minimal memory space with low double-counting bias. When a prevalent byte-sequence shows sufficient dispersion, it is automatically converted into a Snort [61] formatted content-signature that can be used to automatically block connections containing the signature. Earlybird was shown to be extremely effective against the worms present in the wild in 2004, and also efficient enough to be deployable on large networks [66]. In addition, Madhusudan and Lockwood showed that a system based on Earlybird's basic design could be implemented in hardware for even greater scalability [67]. The effectiveness of EarlyBird was further increased through research by Gopolan et al. who developed a more effective suspicious traffic identification algorithm [68]. Instead of simply looking for content prevalence and dispersion when identifying suspicious traffic, Gopolan et al. employ additional metrics that examine the *fanout* of unsuccessful attempts from suspicious hosts. This reduces false positives by avoiding categorizing common protocol headers as malicious.

The PAYL system proposed by Wang and Stolfo [69, 70] detects suspicious traffic by performing a statistical analysis of packet payloads into a protected domain. When packets are observed that don't match the statistical profile of the domain, they are considered suspicious. The statistical profile of the protected domain is measured by first grouping connections by destination port and length — based on the observation that different services will have different profiles, and within a given service, packets of different lengths will have different profiles — and then performing n-gram analysis against each group. The PAYL system uses 1-grams (single bytes), and computes the mean and standard deviation of each

1-gram within a given sliding window size. These means and standard deviations form the feature vector that describes the profile of traffic for that group. Packets are considered suspicious when their Mahalanobis distance from the standard profile for their group exceeds a given threshold. When a host receives a suspicious packet and then begins sending suspicious packets on the same port number, they are considered to be worm packets. A longest common subsequence algorithm is then applied to generate a worm signature which can be used to block all worm traffic. PAYL was enhanced in 2006 with the packets first being classified with a self-organizing map in the POSEIDON system [71]. An advanced version of the PAYL system uses high-order n-grams to thwart content-mimicry attacks [72], but otherwise operates in a similar fashion to the original PAYL system.

Assessment: Whereas dynamically generated signatures can defend against zero-day worms, they are again powerless to stop polymorphic worms.

A greater problem with any sort of dynamic content-based worm detection is that if the content analyzer can be duped into labeling legitimate traffic as worm traffic, the detector itself can be used to disrupt normal traffic. In 2006, Perdisci et al. showed that such detectors could be rendered useless by the careful injection of *noise* packets [47]. This work was extended in [48] to to show the full extent of the threat and formally classify the problem as a learning problem in an adversarial environment. A similar work by Chung and Mok [46] showed that Autograph could be used to constrain legitimate traffic. One proposed solution is to use a corpus of legitimate traffic to filter signatures against, but further research by Chung and Mok showed that even the use of such a corpus will not prevent allergy attacks [73]. This problem also applies content-based detection mechanisms that employ advanced signatures.

Advanced Signatures

One limitation of the above systems is that they all result in a simplistic contiguous fixed signature that matches against any packet that contains it. These signatures are inherently limited, they must be long enough to be specific to the worm content so as not to accidentally block legitimate traffic, yet as short as possible to be as sensitive as possible to the worm. Even simple polymorphism can evade a signature of this type, rendering it useless. Thus a class of detectors with advanced signatures that employ additional information beyond basic byte matching has been developed.

Polygraph is an extension to the Autograph work which enables it to generate a variety of more advanced signatures making it less vulnerable to polymorphic worms [74]. The authors observe that the majority of exploits show invariant code in the exploit framing and overwrite values, and conclude that these invariants can be used to build a successful signature. They enhance autograph to build signatures of three types: conjunction, token subsequence, and Bayes (see Figure 3.6 for examples of conjunction and token subsequence). A conjunction signature matches against content when all of its tokens are found in any order. A token subsequence signature is similar, but only matches when its tokens are found in a specified order. A Bayes signature consists of a series of tokens with corresponding scores. If

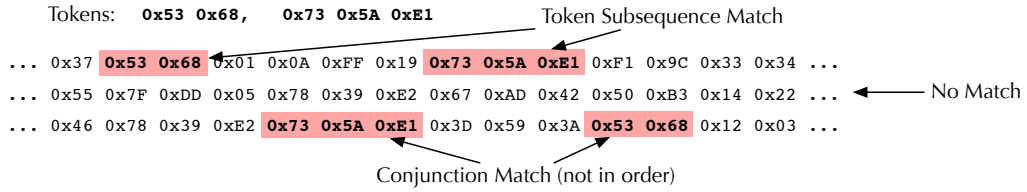


FIGURE 3.6. **PolyGraph's advanced signature** uses tokens in different ways. A Token Subsequence signature matches all tokens in order (Line 1). A conjunction signature matches all tokens in any order.

the content contains enough tokens to exceed an overall threshold, it is considered a match. Polygraph generates these signatures in a manner similar to autograph. It relies on a pool of suspicious connections which it can then analyze. In Polygraph's case, the token extraction is more thorough, extracting all distinct tokens of a minimum size that are not substrings of another token (unless they appear outside of that token with a specified frequency). This gives Polygraph a larger and more comprehensive pool of tokens to match on. Conjunction signatures are generated by finding sets of tokens which exist in every sample in the suspicious pool Token-subsequence signatures are generated by first converting each connection in the suspicious pool to the tokens extracted (separated by whitespace), then iteratively applying the Smith-Waterman string alignment algorithm to find the longest ordered sequence of tokens that matches all connections in the suspicious pool. Bayes signatures are generated by first assuming that each token appears independently from other tokens (whether the string is known to be innocuous or worm), then calculating the percent of connections in the suspicious pool the token appears in and the percent of connections in the innocuous pool the token appears in. For each token, whichever of these percentages is higher is used in the classifier and resulting signature. The overall threshold is calculated based on the observed rates of false positives and false negatives when the signature is applied to the the connections in the suspicious and innocuous flow pools. To generate all three of these types of signatures against a suspicious pool that contains more than one type of worm, the suspicious pool is first grouped by applying hierarchical clustering, and then running the signature generation against each cluster.

The Hamsa system [75] by Li et al. is similar in design to Polygraph. It uses a flow classifier to produce pools of normal and suspicious traffic and generates

signatures from tokens appearing in the suspicious pool. A key distinction in Hamsa is that it provides provable robustness against polymorphism as long as at least minimal fixed byte sequences are present. Hamsa uses a suffix array based algorithm to extract tokens, and generates signatures consisting of multi-sets of tokens. A connection matches a signature if it contains the required tokens with the required frequency (similar to the conjunction signature of Polygraph).

Tang and Chen [60] identify suspicious traffic in their PADS system by employing a double-honeypot which isolates worm traffic from general background noise. They then introduce a position-aware distribution signature system which uses byte frequency distributions combined with simple string matching to produce a signature capable of detecting primitive types of polymorphic worms. A PADS signature of size ω defines the probability of each byte to appear at each location from $0..\omega$ for both legitimate traffic and a specific worm variant. To match some content against a signature of size ω, PADS segments the content into overlapping chunks of size ω and for each chunk finds the location that maximizes the score against the worm probabilities and minimizes the score against the legitimate traffic probabilities. The score for the content is the maximum value for any segment's worm score divided by its legitimate score. If the score for the content exceeds a given threshold, the content is considered to match the worm signature.

Assessment: Dynamically generated signatures advanced enough to catch polymorphic worms would be a truly useful form of worm detection. Unfortunately Crandall et al. [53] showed that Polygraph needs relatively long tokens, and is therefore unable to catch many windows-based worms which require only short fixed payload elements. Hamsa can detect polymorphic worms using with only short fixed content sequences, but still requires those fixed elements to be present.

Furthermore, the recent research into polymorphism by Song et al. [42] has shown that modern polymorphism engines can generate code that very nearly looks like random bytes, posing a significant challenge for signature based systems. Fogla et al. introduced the notion of a blending attack [44], where a worm could control its polymorphism to make it better blend in with legitimate traffic, making it harder for content-based detectors to discern it.

Protocol Analysis

The previous content-based systems all generate *exploit-based* signatures based on invariant content and are therefore susceptible to polymorphism attacks. The LESG system [76], on the other hand, examines network data for the excessively long field lengths which are used to overflow a buffer. It requires an understanding of each protocol it monitors so that it can parse the network data into protocol level fields. LESG considers a field to be a sequence of bytes with special semantic meaning within the designated protocol. Fields can be of fixed or variable length, and in some cases a series of fields may be associated into a single semantic meaning and are therefore concatenated. LESG assumes that all variable length fields are vulnerabilities. Like many of the other content-based systems, it relies on a separate flow classifier to identify suspicious flows, and it then builds signatures by comparing the suspicious and legitimate flows.

Assessment: LESG represents an advance over other content-based systems because it is robust against polymorphic worms while retaining the advantages of network gateway-based deployment. Additionally, LESG is resilient to many of the forms of allergy attacks that plague the other content-based systems. It is limited in that it can only detect attacks in the protocols that it understands (and it is

unclear how quickly new protocols will be decoded), and in that it won't detect attacks that don't exploit a buffer-overflow as the infection vector.

Behavior-based Techniques

In contrast to content-based worm detection techniques, behavior-based worm detection attempts to detect the presence of a worm by monitoring the network without examining the payload of transmitted packets. Instead, these techniques rely solely on patterns of network activity that are characteristic of worm-specific behavior, such as the aggressive scanning a worm might rely on in searching for vulnerable targets.

Like content-based systems, behavior-based systems are typically easily deployable as they are often amenable to being installed at a network gateway. They offer an additional advantage in that they are robust against content polymorphism, thus potentially providing better overall coverage. On the other hand, behavior-based systems typically produce less information gain. They can detect the presence of the worm, and may be able to identify which host is infected, but they typically cannot generate a signature that could be used to block individual worm connections.

There are a number of behaviors exhibited by a worm that may be observed in network traffic. The most obvious is the scanning behavior of a worm as it attempts connections to many destinations to infect new victims. This scanning behavior is the basis several behavioral techniques, each of which perceives it in a slightly different manner.

Connection Failures

Random scans will invariably lead to attempted connections towards *dark addresses* where no host is present. One technique for detecting worms is to watch for the failures that occur when these connections to dark addresses fail. Berk et al. employ this technique in their work, watching for *ICMP Undeliverable* packets indicating connection failures [77]. A worm which is actively scanning will generate higher numbers of these packets than will be observed in normal activity, so a system monitoring the level of ICMP undeliverable packets will be able to detect the presence of a worm. Berk et al. propose having routers forward these messages to a central server where they can be analyzed. A sliding window of ICMP Undeliverable messages is maintained for each source address, and when the number of hosts showing unusually high connection failure rates exceeds some threshold, the alert is raised that a worm is active.

A similar work, the Worm Early Warning (WEW) system by Chen and Ranka uses *TCP reset* messages to the much the same effect [64]. The TCP reset message is sent in response to a syn packet when the target host exists, but does not service the target port. WEW observes the outgoing TCP reset messages at a gateway to determine scan activity inbound from the Internet and potentially the presence of a worm. The gateway intercepts the TCP reset message and replaces it with a SynAck packet, to establish whether the scan source address was spoofed. WEW can then maintain a list of the number of scanning sources, and raise the alert that a worm is present when the number exceeds a given threshold. WEW performs better when the gateway monitors a large, densely-populated address-space, but many small WEW monitors can aggregate their results to achieve the same results.

Rather than attempting to detect worm activity in the Internet, Schecter, Jung, and Berger rely on failed connections to detect worm scanning originating from a protected network. Their system employs reverse sequential hypothesis testing to quickly find hosts with too many connection attempts that fail [35]. For each outgoing TCP syn packet or UDP packet, they consider the connection successful if a synack response is receive or any UPD packet is received within a timeout period. The success or failure of each connection is recorded against the originating host and is evaluated via a reverse sequential hypothesis test. Sequential hypothesis testing takes two hypotheses (the host is infected and the host is uninfected in this case), and considers each event in sequence to see which hypothesis is more likely to be true. When a threshold is exceeded, in one direction or the other, that hypothesis is considered to be true. Sequential hypothesis testing is useful because it provides guarantees on the number of false positives and false negatives, and it minimizes the number of observations needed to make an assessment based on the strength of the evidence observed. In this case, the connections are run in reverse order (hence the reverse sequential hypothesis testing) such that if a host becomes infected and begins scanning, it will be detected as quickly as possible because those scanning connections (the strong evidence) will be considered first. They couple this with a credit-based connection-rate-limiting feature that prevents high-speed worms from quickly scanning large numbers of addresses.

This technique is also employed by Autograph [65] to establish a pool of suspicious connections to be further evaluated by its content-based analysis engine.

A 2006 work by Cheetancheri et al. [78] relies on connection failures as well, but has each host determine its own status of infected or not. Hosts share their

status with their peers. When a host receives a message from another host saying it is infected, it passes it along with its own status information attached. When many hosts are infected, the group can come to a consensus that a worm is actually present.

Assessment: Those systems that rely on detecting connection failures ([77, 64, 35]) attempt to detect the worm as it searches for new targets. Connection failures occur when the worm attempts to connect to a dark address or a to a service (or port) that isn't provided by the remote host. This technique suffers from a couple of significant drawbacks. The first is that there may be no way to determine whether a connection is successful or not. TCP connection failures are easy to determine, they fail when the three-way handshake specified in the protocol doesn't complete within a timeout period. UDP connections, on the other hand, have no notion of a handshake, and there is no clear way to identify whether they are successful or not. One metric is to say that a UDP connection fails if there is no UDP traffic returned from the destination back to the source, but this relies on applications actually behaving this way which may not happen in practice. Another drawback to this technique is that it relies on the worm making connections to addresses that only have a low probability of existing. If a worm has access to some other means of identifying which hosts are likely to exist — perhaps examining web browsing history or peer-to-peer network interaction logs in the case of a topological worm, or simply having a pre-computed hit list — it won't show a high connection failure rate and this technique won't detect it. This lack of coverage is a significant limitation of connection failure techniques.

Network Telescopes

The three systems above rely on the random scanning nature of worms to generate connection failures which can be observed. However it is also possible to observe this behavior simply by monitoring large numbers of addresses that should not receive traffic, in what is called a *network telescope*. Network telescopes enable one to observe the aggregate scanning behavior of hosts across the entire Internet.

Zou et al. were among the first to use this idea for worm detection [6]. They proposed using distributed ingress monitors to watch for connections to dark addresses (in a way very similar to the work by Berk et al. above) coupled with egress monitors that could capture the scanning behavior of a host once it was deemed to be infected. The monitors would report to a central observatory which would analyze the infection and scanning rates, attempting to build a model for the worm propagation. When the overall scanning rate exceeded a given threshold, the system would begin to employ a Kalman filter on the number of scans to determine whether the effect was a new worm or simply an overall increase in scanning activity.

Wu et al. proposed a similar solution [7], again with distributed detectors monitoring connections to unused addresses and forwarding their results to a central observatory. They consider a host to be infected if it makes at least two scans to unused addresses. An adaptive threshold on the number of hosts considered infected accounts for noise and helps to prevent false positive reports. When the number of infected hosts increases between time ticks by more than some value, an alert is raised that a worm is active.

This usage of a network telescope was enhanced and refined by Bu et al. [79] who observed that scan arrival times appear as a non-stationary poisson

process rather than the gaussian distribution assumed above, and that this allows a more sensitive detector. They use a two stage detector. The first stage employs a CUSUM procedure that triggers the second stage when scan inter-arrival times show an exponential pattern of decrease. The second stage determines whether the decrease is due to a worm attack and estimates the worm propagation model.

Yegneswaran, Barford, and Plonka showed the practicality of a network telescope based solution by actually implementing and testing one against real networks [80]. They used their system to monitor roughly 16 million addresses and found that it successfully detected the LovGate email worm.

Finally, Rajab, Monrose, and Terzis furthered the practicality of such systems by analyzing the coverage required to detect worms given the non-uniform distribution of hosts in the Internet [81]. They showed that many small monitors are more effective than a single large monitor, and when the monitors are placed in the most populated prefixes, they are even more effective.

Assessment: Similar to the connection-failure technique described above, the network telescope technique employed by some systems [6, 7, 80] is fragile in that it relies on somewhat random scanning by the worm. A topological worm like the Santy worm won't scan dark addresses and won't be detected by this technique. Additionally, Rajab, Monrose, and Terzis showed that even if a worm is not topologically scanning and doesn't have a hit-list, it can limit its exposure to network telescopes via careful scanning [45]. They showed that a worm can efficiently scan address blocks with probes designed to blend into background noise, to determine whether there are active hosts within the address block.

Pattern of Destination Addresses

The network-telescope and connection-failure schemes outlined above rely on the worm scanning randomly, causing it to attempt to connect to dark addresses. However, it is also possible to detect scanning behavior without depending on the scanning to be random in nature. The enabling observation is that during normal operation a host shows a pronounced pattern in the number of unique destinations it connects to that differs from the pattern exhibited when scanning.

The SWORD system [82] relies on this fact as one of its components in worm detection. SWORD measures the distribution of the number of visits each destination receives versus the rank ordering (by popularity) of the destination. An uninfected host exhibits a distribution that resembles a power-law while an infected host that is actively scanning exhibits a flat distribution. These can be distinguished by doing a linear least-squares regression analysis on the log of the number of visits versus the log of the popularity rank. When the correlation coefficient is weak (close to zero), it indicates that the host is actively scanning. The threshold that distinguishes infected from uninfected is based on training the detector on legitimate traffic.

Sekar et al. measure this behavior in a slightly different way [83]. They observe that the number of unique destinations visited by a normal host grows sub-linearly with the size of the history, and that as the history length increases, the rate of increase in unique destinations slows. By employing multiple threshold/history size pairs, they can catch fast-scanning worms early with a short window, but also catch slow-scanning worms with a longer window. To choose these history sizes and thresholds, a choice must be made between detection latency and the false positive rate. They present an Integer Linear Programming solution for

choosing history-sizes and thresholds based such that the detection latency and false positive rate are minimized based on training data from legitimate traffic.

Jung et al. extend their use of sequential hypothesis testing introduced in the TRW paper [84] to test the rate with which first-contact connections are made in their RBS+TRW work [85]. They observe that the inter-arrival time of first-contact connections follows an exponential pattern. They employ sequential hypothesis testing against new connections observing whether each connection fits that model or not. This procedure is adaptive to bursty legitimate connections. They then combine this work with their TRW work to produce an effective behavior-based worm detector.

A related technique is employed by EarlyBird [66] to identify suspicious traffic. A scanning worm will send its attack payload to many destinations, so EarlyBird looks for identical content that is widely dispersed and adds that traffic to its suspicious pool, to be passed on to its content-based analysis engine for further study.

Assessment: The destination address based techniques provide better coverage than the connection failure techniques discussed previously, as the worm simply cannot avoid contacting destinations if it wishes to propagate. Worms that wish to propagate widely must connect to many destinations, and this scanning behavior can be detected, regardless of the scanning mechanism. The goal of a worm is to propagate widely, and to do so the worm must connect to many destinations. Even if a worm relies on a hit-list or topological information, every connection it makes that is not piggybacked on a legitimate connection changes the pattern of the destinations addresses visited. Three existing works have used this technique successfully [82, 83, 85], and while it remains an open research

question the extent to which the worm can blend its connections with the stream of legitimate ones, it is clear that any worm action must at some level change the pattern of destinations visited.

Causation

The scanning behavior of worms is triggered by being infected by a connection from another infected host. This idea can be thought of in the context of one connection *causing* another connection to occur. It was first used in worm detection in the GrIDS work published in 1996 [86]. GrIDS employs LAN specific monitoring nodes that build graphs of network activity. A worm is identified by the distinctive tree shape it forms, as a single infected node infects several neighbors, who in turn infect several of their neighbors. GrIDS employs a sophisticated rule-based scheme for graph building, where as each connection is reported, it may be added to one or many graphs based on sharing hosts and temporal proximity with the existing graphs. Each LAN then shares it information with its parent network, but only in a condensed form (i.e. the local network's graph is condensed to a single node in the parent network's graph). This graph reduction was intended to decrease overhead, but also has the impact of decreasing the information available to the parent node.

The ideas behind GrIDS were extended in 2002 by Toth and Kruegel [87]. They observed that the incoming infection vector to a host is typically the same one that the worm scans for in its outgoing connections, and that this content similarity could be leveraged to further enhance the effectiveness of the detector. They further adopted connection failures as an additional indicator and require significantly less overhead than GrIDS. Toth and Kruegel's detector works by

building host-specific connection histories. A connection history contains all of the connections sent to that host, along with a portion of the payload. Connection trails for each host are also maintained, which represent chains of connections that end at the host, where a chain of connections consists of a series of connections ordered in time where the destination of one connection is the source of the next connection, the final destination in the chain is the host in question. An outgoing connection from this host is compared with the incoming connections in the connection history and connection trail. When similar payloads are found, the connection is added to a suspicious pool. Each time the suspicious pool for a host is updated, the number of repeating elements (connection payloads) is counted, as is the number of connections to non-existent hosts and non-existent services (referred to as obsolete connections in the paper). These three counts are weighted by factors established by the network operator, and if the sum of the weighted counts exceeds a given threshold, the host is considered to be infected.

In 2004, Ellis et al. presented a similar approach with a more in depth analysis [88]. Their behavioral worm detection work relied on three causal identifiers to detect worms: a server changing into a client, content similarity between incoming and outgoing connections from a host, or a link predicate that may only be useful when considered in the context of the connection graph (such as connections that aren't closed properly). They observe that in many network architectures, a server will never initiate connections, but limits its activity to responding to connections initiated towards it. A host that changes this pattern is likely infected with a worm. Their second worm identifier is similar to an aspect of Toth and Kruegel's work (which was not cited in the work by Ellis et al.). They note that the connection that infects a host will cause outbound connections with

the same payload (ignoring the potential of polymorphic worms), and observing this similarity is a good indicator of causality. Their final identifier, is the notion of link predicates that identify worm connections. These predicates could range in specificity from: host a contacted host b to a detailed signature such as host a sent a UDP packet to host b on port 1434 with the contents equal to the SQL Slammer worm. They suggest that a middle ground such as: host a opened a TCP connection to host b and didn't close it gracefully might make a nice middle ground of sensitivity and specificity. Once this link predicate is identified, it can be used to build descendant relationships in the connection graph (where two hosts are connected by a link that satisfies the link predicate) forming suspicious subgraphs that can be analyzed. The suspicious subgraph can have the following attributes measured: depth, number of descendants, branching factor, and time to get to a specific depth. If these attributes match specific values, such as a branching factor of greater than one, that can indicate the presence of a worm. Unfortunately no detailed analysis was performed of specific link predicates or attribute thresholds.

SWORD (discussed above [82]), employs yet another take on causality as one of its components for detecting worms. SWORD maintains a causal connection graph, where each node in the graph represents a single connection and the directional links represent potential causality. A new node (connection) added to graph becomes a child of existing nodes in the graph that satisfy the Lamport "happened-before" constraint and are potential causes of this new node. Connections to the same host that originates the new connection are potential causes, as are outbound connections from the originating host. To keep the graph and node degree manageable, causality is transitive, such that an ancestor connection that is potentially causal of a parent node, is also potentially causal

of the child node. When a node is added to the graph, it is compared with its ancestors for similarity, where similarity is defined as having matching connection attributes among: protocol, destination port, and TCP flags. Unlike the above works [88, 87], content is not considered as an attribute for similarity to maintain robustness against polymorphic worms. When the number of similar ancestor connections exceeds a threshold established during a training process, then the child connection is considered suspicious. If a connection is considered suspicious by this heuristic as well as the destination address technique described above, it is tracked in a sliding window. When sufficient suspicious connections are present in the window the alert is raised that a worm is present.

The DSC system by Gu et al. is another work that seeks to identify transitions between server and client [37]. They consider it suspicious when a host receives a connection to a given port and then begins making outgoing connections on that same port. This work does not rely on connection graphs, maintaining only limited state information for each host. A training period establishes normal connection rate parameters for each host in the network. After a host receives an inbound connection on a given port, its rate of outbound connections to that port is monitored and an alert is raised if it exceeds the bounds established during training. Once a host is identified as suspicious its outgoing connection rate is monitored. If the connection rate is above a threshold for that host that was established in a training period, then the host is considered to be infected.

Assessment: Ellis et al. identified three types of network application architectures: pure client-server, where no host may act as both a client and a server; client-server, with a similar restriction but loosed such that no host may act as both a client and a server for the *same service*; and ad-hoc networks

that have no such restrictions. They observed that some causal identifiers like a server turning into a client are extremely effective in the pure client-server and client-server architectures, but entirely ineffective in ad-hoc networks, which dominate academic environments.

The causation technique at first glance appears to be extremely effective, what better way to detect a worm that to observe the connection that infected it and the resulting outgoing scans. However, there are some limiting factors. A worm may employ multiple attack vectors, making it difficult to correlate the incoming connection and with the resulting outgoing connections because they would share no similarity. Additionally, worm detectors placed at a network gateway would only be able to observe infections from the external network and the resulting outgoing scans. If a host was infected internally, the infection connection would not be visible to the monitor making it impossible to correlate with the outgoing connections. For this reason, techniques that do not rely on observing the infecting connection will be more reliable and comprehensive.

Graph-based Detection

Many worm detection techniques have been refined over the years, as both the ideas of what a "normal" network is and the expected behavior from a worm have changed. One interesting example of this is graph-based detection, (which is included *causation* category in this work). This is one of the oldest detection techniques. It was first presented in a work by Staniford-chen et al. in 1996 [86] but is still undergoing refinements to the basic idea.

The initial idea as proposed in GrIDS relied on a heavyweight monitoring solution and a rigid hierarchical structure controlled by a central *organization*

hierarchy server, or OHS. It appears that the authors expected networks to be well controlled and show predictable behavior.

The works by both Toth and Kruegel [87], and Ellis et al. [88] relaxed the requirements for this rigid structure while providing more context to the causal tree by looking at network payload and link predicate rules. This had the effect of reducing the overall deployment overhead while increasing the accuracy of the worm detectors. In both cases however, worm detection relies on having global network knowledge and on observing the payload. Both works suggested that a gateway node in promiscuous mode would collect all network traffic, but that is not that case in a large network. Global network knowledge is difficult to come by in networks comprised of more than one subnet, as traffic within a subnet won't make it to a root monitoring node without dedicated hardware to route it there. This may make such systems impractical to deploy. Furthermore, polymorphic worms will not show the payload similarity that these systems rely on, rendering them unable to detect an entire class of worms.

These developments led to the SWORD system, which further relaxes the constraints on causation. Because SWORD will observe only that traffic which crosses the gateway where it is deployed, it is unable to reconstruct an entire infection graph to look for tree structures. The threat from polymorphic worms makes strongly identifying causation (incoming content matching outgoing content) impossible. These two factors lead SWORD to rely on weak potential causation, which by itself isn't accurate enough to detect worms. Only when coupled with complementary heuristics does this causal connection graph technique work effectively.

This evolution of graph based detection from a rigid structure to a much

looser structure in many ways mirrors the evolution of the network as a whole. Systems like DHCP and open wireless networks have transitioned the typical network from a centrally allocated and controlled network to a more free flowing and ad-hoc design. Worm detection techniques must adapt to these changes.

3.3. Detection Systems

Having discussed the individual worm detection techniques in isolation, we now turn our attention to complete worm detection systems and their overall performance. Table 3.2 lists a selection of published worm detection systems and which techniques each system employs. We will examine the systems with regards to each of the desired worm detection attributes discussed in Section 3.1.

Speed and Accuracy

One of the most difficult aspects of comparing worm detection systems is comparing their performance. There are no standardized tests that are run on all detection systems, and even the metrics used vary widely. In Table 3.3 we have summarized the performance data culled from the papers which present these systems. Frequently the performance of a system is best represented not with a single number, but with a ROC curve, comparing the sensitivity to specificity of a system over a range of configuration options. However, virtually no systems present their results this way. Many systems, in fact, are presented without any supporting evaluation.

TABLE 3.2. **Worm detection techniques (in order of publication year)**

	Host-based			*Honeypot*	*Content-based*				*Behavior-based*			
	Buffer Overflow	Input Correlation	System Calls	Honeypot	Static Signature	Dynamic Signature	Adv. Signatures	Protocol Fields	Network Telescope	Conn. Failures	Address Distribution	Causation
Snort [61]					•							
HoneyComb [58]				•		•						•
NetBait [63]					•							
Berk et al. [77]										•		
Zou et al. [6]									•			
Wu et al. [7]									•			
Autograph [65]						•				•		
Reverse SHT [35]						•				•		
EarlyBird [66]						•					•	
HoneyStat [51]	•			•								•
iSink [80]									•			
DSC [37]												•
PAYL [69, 70]							•					
HBD [89]									•			
COVERS [49]	•	•										
WEW [64]				•		•				•		
TaintCheck [52]		•										
DACODA [53]		•										
Vigilante [55]		•										
Malan et al. [56, 57]			•									
Polygraph [74]							•					
PADS [60]							•					•
Bu et al. [79]									•			
Hamsa [75]							•					
Anagram [72]							•					
SWORD [82]											•	•
HonIDS [59]				•		•						•
COOP [78]										•		
MRW [83]											•	
Poseidon [71]							•					
Brumley et al. [90]							•					
Sweeper [50]	•	•										
LESG [76]								•				
d-ACTM [91]										•		
TRW+RBS [85]										•	•	

TABLE 3.3. **Accuracy and speed of selected systems**

System	**Accuracy**	**Speed**
COVERS [49]	No measured F-, F+ rated as unlikely	Detects worm on first attack connection, <10ms for signature
TaintCheck [52]	F+ in 2 of 15 experiments, no F-, Generated signatures show F+ 0.0015%	-
DACODA [53]	-	-
Vigilante [55]	-	<400ms to deploy filter on vuln. host
Sweeper [50]	-	<60ms to filter
Malan et al. [56, 57]	-	5 seconds
HoneyStat [51]	No measured F+	-
HoneyComb [58]	-	-
HonIDS [59]	-	-
Snort [61]	Dependant on Signatures	-
Autograph [65]	Best case of 0 F+ and 0 F-	<1% of hosts infected (with distributed monitoring)
EarlyBird [66]	F+ yes, F- unknown but none reported vs Snort signatures	-
PAYL [69, 70]	-	-
Polygraph [74]	F+ <0.01% for most cases, no F-	Variable
PADS [60]	-	-
Hamsa [75]	F- 0%, F+ 0.1839%	64-361 times faster than Polygraph
POSEIDON [71]	F- 26.8%, F+ < 1%	-
Anagram [72]	F- 0%, F+ 0.006%	-
LESG [76]	F- 0%, F+ 0% for most scenarios	Signature generation <5 seconds
Berk et al. [77]	-	5 seconds (4 hosts)
WEW [64]	-	<5 hours for Code-Red worm

Continued on next page

Table 3.3 – *Continued*

System	**Accuracy**	**Speed**
Reverse SHT [35]	Efficiency 0.324, Effectiveness 0.917	<10 first-contact connections
Cheetancheri et al. [78]	-	14 seconds, 32% of vuln. hosts infected
Zou et al. [6]	-	1-2% of vuln. hosts infected
Wu et al. [7]	-	<1.5% of vuln. hosts infected
Bu et al. [79]	-	<10 % of vuln. hosts infected
Yegneswaran et al. [80]	-	-
SWORD [82]	F- 0% for random/local scanning, 10% for topo scanning, F+ 0%	<10 seconds for 100 scans/sec, <250 seconds for 1 scan/sec
Multi-resolution [83]	F+ 0.04 per 10 seconds, F- 0%	<500 seconds for all worm types
DSC [37]	-	<0.64% of vuln. hosts infected
Dubendorfer et al. [89]	-	-
TRW+RBS [85]	<1 F+ per hour, 2 measured F-	7 first-contact connections
d-ACTM [91]	F+ 0.1% vs silent worm	<7% of internal hosts infected by silent worm

Coverage

The coverage of a system is the percent of different types of attacks that it can detect. Most published worm detection proposals discuss the coverage of the proposed system at least in passing, but few devote any great level of detail towards it. Furthermore, a review of the published literature has revealed that the coverage of worm detection systems is directly tied to the techniques it employs. To

discuss coverage then, we present not a list of the systems as we did in the section on performance, but instead list the detection techniques with analytically derived descriptions of each technique's coverage in Table 3.4)

Table 3.4 shows that host-based techniques (Buffer Overflow, Input Correlation, and Systems Calls) have good coverage, with the exception of application-level attacks. Honeypot-based systems have good coverage against untargeted worms, but only for those applications running on the honeypot. The content-based techniques have good coverage except against polymorphic worms. It remains to be seen whether advanced signatures can detect the most polymorphic of worms. Behavior-based techiques have good coverage against high-speed and random scanning worms, but will fail to be effective against slow-scanning, targeted worms.

Capabilites

Different worm detection systems have different capabilities. Some are capable of detecting simply that a worm is present within the network, while others are capable of detecting which individual network connections contain the attack, what the vulnerability is, and which hosts within the the network are infected Table 3.5 lists the various capabilities in the left-hand column, and the worm detection systems that possess that capability in the right-hand column. The capabilities are generally equivalent for systems using the same detection techniques.

Systems using content-based detection techniques are typically the most

TABLE 3.4. **Detection technique coverage**

Technique	**Coverage**
Buffer Overflow	Misses application-level injection attacks, otherwise good
Input Correlation	Misses application-level injection attacks, otherwise good
System Calls	Good
Honeypot	May miss targeted scans, misses attacks against unmonitored applications
Static Signature	Misses zero-day worms for which signatures have not yet been developed
Dynamic Signature	Misses polymorphic worms
Advanced Signature	May miss advanced polymorphic worms
Conn. Failures	Misses targeted scans if they experience low connection failure rates
Network Telescope	Misses targeted scans (like topographical worms)
Causation	Good for single attack vector worms
Dest. Addr. Dist.	Good for worms with scan rates greater than normal traffic

capable, able to develop an attack specific signature and detect all actively scanning hosts. Host-based and Honeypot-based techniques are nearly as capable, but are only able to detect those hosts that actually attack a protected host. Systems built using behavior-based techniques are typically the most limited in capabilities. Some, such as TRW+RBS can detect all active infected hosts, but many behavior-based systems can do no more than simply detect the presence of a worm in the network.

Analysis of Detector Systems

Table 3.6 outlines the major pros and cons of each type of worm detector.

Host-level techniques that observe memory accesses to detect buffer overflows

TABLE 3.5. **Worm detector capabilities**

Capability	**Systems with this capability**
Detect that a worm is present	Berk et al. [77], WEW [64], Cheetancheri et al. [78], Zou et al. [6], Wu et al. [7], Bu et al. [79], Yegneswaran et al. [80], Dubendorfer et al. [89], Malan et al. [56, 57]
Detect which hosts in a protected network are infected	Reverse SHT [35], SWORD [82], Multi-resolution [83], DSC [37], TRW+RBS [85], Snort [61], Autograph [65], EarlyBird [66], PAYL [69, 70], Polygraph [74], PADS [60], Hamsa [75]
Detect attacks against specific hosts (protected hosts or honeypots)	COVERS [49], TaintCheck [52], DACODA [53], Vigilante [55], Sweeper [50], HoneyStat [51], HoneyComb [58], HonIDS [59]
Generate attack specific signature	COVERS [49], TaintCheck [52], DACODA [53], Vigilante [55], Sweeper [50]
Generate content-based signature	Autograph [65], EarlyBird [66], PAYL [69, 70], Polygraph [74], PADS [60], Hamsa [75]

are effective against the class of worms that exploit buffer overflows, but they impose a performance cost on all operations on the computer. Additionally, unless they are built into the operating system host-level detection systems pose a substantial deployment overhead for large organizations.

Honeypot-based techniques have limited coverage in that they may not detect hit-list or topological worms.

Content-based techniques have the advantage that they generates signatures that can be used to block worm traffic, and that they are easily deployed: requiring only a single monitor for a given organization. These advantages are offset by their inability to detect polymorphic worms, and their susceptibility to malicious training attacks.

Behavior-based worm detection systems do not have the weaknesses shown by content-based systems. They are robust to polymorphic worms; and as they do not generate signatures used to block traffic, they can't be duped into blocking legitimate traffic. However, current behavior-based systems provide limited information, they may tell you that a worm is present, but most likely can't tell you what exploit is being targeted. Beyond this limitation, many of the more naive content-agnostic detectors can be easily avoided by intelligent worm design.

3.4. Detector Selection

Having established the pros and cons of different worm detection mechanisms, we can select some algorithms as candidates for further study. We performed an

TABLE 3.6. **Pros and cons of categories of worm detection**

Category	**Pros**	**Cons**
Host-based	▸Robust against Polymorphism	▸Per-machine deployment ▸Buffer-overflow attacks only
Content-based	▸Deployable at gateway ▸Fast vs non-polymorphic worms	▸Vulnerable to polymorphism
Behavior-based	▸Deployable at gateway ▸Robust against polymorphism	▸No signature generation

extensive evaluation of proposed worm detectors, considering 36 different published works. We grouped them into the following categories based on their detection algorithm: host-based detectors, content-based detectors, and behavior-based detectors. Each category has its own strengths and weaknesses.

Detectors that we classified as *host-based* included, among others: COVERS [49], DACODA [53], TaintCheck [52], and Sweeper [50]. COVERS attempts to determine the root cause of a buffer overflow by forensic analysis of the system memory. DACODA uses full-system symbolic execution to track data from the network and observe malicious memory manipulations. TaintCheck labels untrusted data as tainted, then tracks its propagation through memory and monitor its usage. SWEEPER combines several of these techniques with a more light-weight monitoring framework. These detectors are very effective and ongoing research continues to reduce runtime overhead. Several factors, however, lead us to

exclude host-based detectors from this study. Host-based detectors require end-host deployment but a network operator may have no control over what software is installed on the end-hosts running in their network. Furthermore, users may circumvent host-based software installs as illustrated by IKEE.B, which targeted only those users who intentionally installed an unsupported operating system. Finally, it is unclear whether host-based systems are capable of detecting an attack like that used by IKEE.B. The systems listed above all rely on observing malicious memory manipulations such as buffer overflows, but IKEE.B did not perform any illegal memory operations; it merely exploited a configuration vulnerability.

Detectors that monitor the network instead of end-hosts seem much more promising because they do not require deployment on each host to be protected. We first look at detectors that examine the contents of network traffic, including AutoGraph [65], EarlyBird [66], PAYL [69], Anagram [72], and LESG [76]. AutoGraph identifies suspicious traffic based on flow patterns, then examines that traffic for frequently occurring byte sequences. EarlyBird operates similarly, identifying prevalent and highly dispersed traffic as suspicious and then counting recurring data elements. PAYL and Anagram take a slightly different approach, performing statistical analyses of packet payloads. LESG identifies the protocol of monitored traffic and looks for long fields that violate the protocol definition. Each of these detection mechanisms share a similar limitation that leads us to exclude them from our comparison: *they are unable to monitor encrypted*

traffic. Encrypted traffic is a special case of polymorphic traffic. Content-based systems designed to catch polymorphic worms (such as Polygraph [74]) depend on attack-specific, invariant sections of content which may not be present for an encrypted worm. Even when worms are transmitted using unencrypted connections, advances in polymorphism research such as [92] have threatened the promise of these detectors. As a further consideration, it is prohibitively difficult to acquire a variety of network traces which contain full network content making it is infeasible to evaluate these detectors.

The remaining and largest class of detectors is behavior-based (or payload oblivious) detectors. These include TRW [35], RBS [93], PGD [94], and many others. These systems also monitor network traffic, but they examine the behavior of traffic from end hosts rather than the contents of their packets. This type of system is easily deployed, requiring as little as a single monitor at the network gateway. They are capable of detecting worms regardless of the scanning mechanism or propagation type (including propagation via encrypted channels), and many of them are capable of identifying the worm-infected hosts. However, we do exclude some behavior-based systems that a network operator could not easily deploy. For example, detectors using network telescopes (such as those by Wu et al. [7] and Zou et al. [95]) require a large dark address space and cannot be deployed by a network operator unless they control a large address space.

After our exhaustive evaluation of worm detectors, we are left with the

following selections: TRW [35], RBS [93], TRWRBS [93], PGD [94], DSC [37], and MRW [83].

These six detectors seem worth of further examination. To this end, we first develop a framework that allows for the easy evaluation of behavior-based detectors across a range of scenarios. This framework is described in Chapter IV. Chapter V discusses the selected detectors in more detail and contains a complete evaluation of their performance characteristics in detecting worms.

CHAPTER IV

EVALUATION FRAMEWORK FOR WORM DETECTORS

In order to compare the worm detectors that we identified in the previous chapter, we need to see how well they detect worms. The best way to do this is to run them against known worm traffic in a variety of different settings and measure their performance. Repeatable controlled experiments that allow us to measure their performance in a variety of situations will enable us to determine which detector would be the most effective in the real world. We discuss the results of this evaluation in the next chapter, but first we present the framework we developed to enable this measurement.

There are four primary components to the framework, as shown in Figure 4.1. The first component processes packet level network traces into our flow-level database format. The second component extracts a subset of the trace that forms a complete *environment* to run experiments in. The third component simulates worm traffic in a given environment. The final component uses the environment and the worm traffic to evaluate the performance of the various worm detection heuristics.

We will now examine each of these components in more detail.

4.1. Trace Conversion

The primary external input to our framework is network traces captured at facilities from around the world. These traces capture real network traffic and

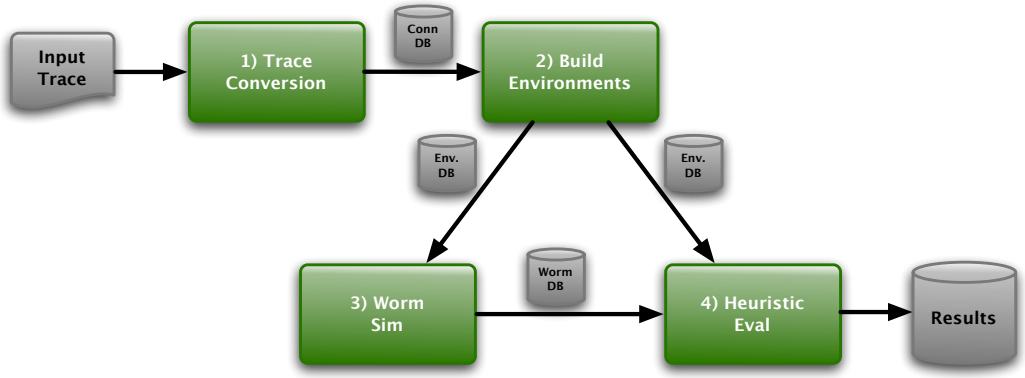


FIGURE 4.1. **Trace conversion architecture**

provide the context that ensures that our experiments provide accurate results. Network traffic from different locations can exhibit very different characteristics, so to fully test worm detectors it is important to test them against as many network traces as possible. The more different network traces we can use, the more confidence we will have in our results, so a key consideration in our framework is supporting as many network trace formats as possible.

The first stage in our framework accomplishes this goal by translating input network traces into a common format. This TraceConversion process is built using a pluggable python architecture that allows new formats to be added easily. The current implementation includes three input adapters: a libpcap adapter that reads tcpdump captures, a CoralReef adapter that can convert any format that the CoralReef toolkit supports (which includes a wide variety of capture standards), and a special converter for the custom binary format that our PKU trace was delivered in.

Our goal is to convert the packet level traces into a format that we can

efficiently and effectively work with. All of the detectors we evaluate are flow level detectors. That is to say, they do not rely on the packet level details of network traffic but only on higher level attributes. When we started this project, the terminology around network "flows" was in its early stages and had not settled into a standard. We named our output data structure a *ConnectionDescriptor*, but today a more common term would be a network *flow*. We use these two terms interchangeably.

The idea of a network *flow* was originally established by Cisco and other router vendors but is not a well defined standard. This means that our definition may not match perfectly with the definition of flows in other contexts. The term "network flow" generally refers to the aggregate network traffic between two hosts on a single protocol. For TCP based traffic, a flow typically represents one complete TCP connection from setup through tear-down. However, we want to refer to the idea of a flow more broadly than that, specifically, we want to include UDP traffic. UDP does not include any connection semantics, so we must establish our own. This requires us to answer some questions, such as: what time delay between packets terminates one flow and begins a new flow? Table 4.1 specifies our flow description, and Table 4.2 describes the data elements we capture about each flow.

The process for converting packets into our common format is outlined in Figure 4.2. Trace converters implement a single method "parsePacket"

TABLE 4.1. **Flow definition**

Flows are defined as a collection of packets between two IP addresses that occur with no more than one minute elapsed between packets and meet the following additional criteria:
For UDP traffic (1) specify the same source and destination port
For TCP traffic (1) meet the UDP flow criteria above (2) are contained within a single TCP connection. If the connection is terminated with fin packets and then a new connection is established with syn packets on the same ports within the timeout period, this new TCP connection is recorded as a new flow
For other protocols We currently disregard communication on protocols other than TCP, UDP, or ICMP

which takes a single line of input and returns the packet data encoded in that line as an instance of our internal Packet class. The packet is handed to the ConnectionBuilder component which maintains a list of currently open flows. If the packet is part of an open flow, the flow is updated with the information from the packet, otherwise a new flow is created. Periodically (based on the observed timestamps) the open flows are filtered for flow that should be closed due to timeout. Closed or timed out flows are written to the database.

The advantage of writing flows to a database instead of a flat file is that we can use the data in different ways. For example, a worm detector that relies only on the opening packet of a connection can sort the connections by first packet time. Conversely, a worm detector that looks at whether a connection succeeds or fails cannot process a connection until that has been established. We record

TABLE 4.2. **Data elements captured about each flow**

Attribute Name	**Description**
startTime	timestamp of first packet in this flow
statusTime	timestamp when we determined the status of the flow
endTime	timestamp of the last packet in the flow
protocol	transport protocol (tcp, udp, icmp, etc.)
sourceAddr	IP address of source as dotted quad
sourceAddrInt	IP address of source as unsigned integer
sourcePort	source port number (TCP and UDP protocols)
destAddr	IP address of destination as dotted quad
destAddrInt	IP address of destination as unsigned integer
destPort	destination port number (TCP and UDP protocols)
packetsSent	number of packets sent from originator
packetsReceived	number of packets received by originator
bytesSent	number of bytes sent by originator
bytesReceived	number of bytes received by originator
syn	(TCP) boolean: was syn packet seen?
synAck	(TCP) boolean: was synAck packet seen?
est	(TCP) boolean: was 3-way handshake completed?
sourceFin	(TCP) boolean: did source send fin packet?
ackSourceFin	(TCP) boolean: was source fin packet ack-ed?
destFin	(TCP) boolean: did destination send fin packet?
ackDestFin	(TCP) boolean: was destination fin packet ack-ed?
rst	(TCP) boolean: was a reset packet seen?
finInit	(TCP) conn. terminated by source, dest, or timeout
pushCount	(TCP) num. of packets observed push flag set
urgentCount	(TCP) num. of packets observed with urgent flag set
ecnEchoCount	(TCP) num. of packets observed ecn flag set
winReducedCount	(TCP) num. of packets with win reduced flag set

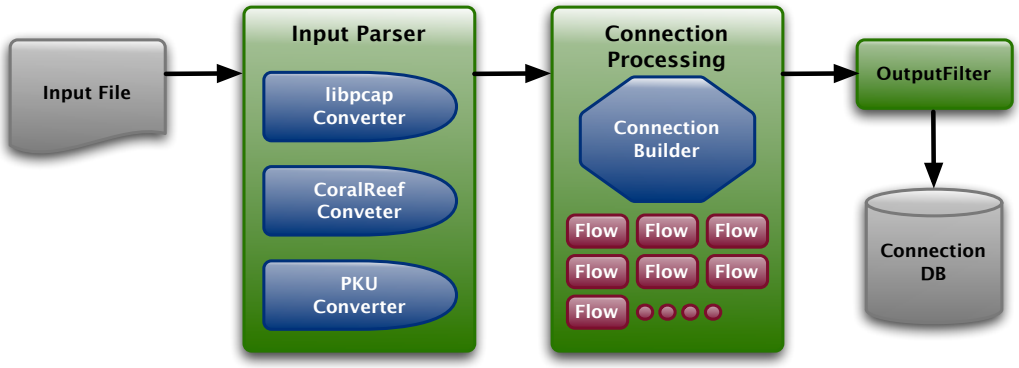


FIGURE 4.2. **Connection extraction process**

this time information as well (as statusTime) and can sort the connections based on that time simply by specifying a different "order by" clause when we fetch the connections.

4.2. Environment Generation

The full traces converted in step one of our evaluation process may contain a larger network or a longer trace than is needed for our evaluations. Additionally, the worm simulation and detector evaluation steps of our process require metadata about the trace they are operating on. The next step in the process then is to build an evaluation *environment* based on the full trace. An evaluation environment consists of an XML descriptor, a trace to perform training on, and a trace to evaluate against. The XML descriptor contains file references to the training and evaluation traces, the address space of the network in the trace, a list of the active internal hosts addresses, and lists of the busiest, least busy, and randomly selected internal hosts.

Building a standard descriptor of a given trace allows us to easily plug different configurations into the worm simulator and detector evaluator.

4.3. Worm Simulation

To generate the worm traffic required to evaluate the effectiveness of the various detectors, we implemented the worm emulator described in [96].

GLOWS takes an *environment* as an input, and simulates the outbreak of a worm in the protected network. GLOWS cannot, of course, simulate the entire Internet. Instead it uses a finite-state model to simulate the behavior of every address in the protected network and every vulnerable host in the Internet; and uses a probabilistic model for the remainder of the Internet. Modeling the vulnerable hosts with the finite-state model yields greater verisimilitude than a pure probabilistic model. It is feasible because the number of such hosts is only a small fraction of the total hosts in the Internet. Furthermore, we don't model packet level interactions which reduces computational overhead and we don't model congestion effects or background traffic.

Ignoring congestion effects impacts the accuracy of the model, but we focus our evaluation at the first stages of worm infection and at that point congestion effects due to the worm should be very small.

GLOWS implements the following worm scanning algorithms as described by Staniford et al. [8]: random-scan, local-preference scan, and topological scan.

A random scanning worm simply chooses target addresses at random from

the entire IPv4 address space. This typically results in many connection attempts to addresses with no host present or with a host that is not running the requested service, resulting in many connection failures. Permutation and sequential scanning worms should show very similar characteristics and are not evaluated separately here.

A local-preference worm scans local addresses (in the same prefix) more frequently than addresses in the full address space. This results in more scans that do not cross the network border (and are therefore not visible to a border-located detection mechanism). Existing local-preference scanning worms, such as Code-Red II [11], target the local /16 network (that is, those network addresses sharing the same 16 high-order bits) approximately 50% of the time, the local /8 (as before, network addresses sharing the same 8 high-order bits) 25% of the time, and the entire network the remaining time. As all our traces are about a /22 network, such a worm would largely resemble a random scanning worm. Instead, our local-preference worm scans the local /22 50% of the time, the local /8 25% of the time, and the entire network the remaining time. This means that the local-preference worm generates fewer connections visible to the detectors and has a much higher chance of infecting multiple targets within the network.

The *topological-scanning* worm starts with a list of addresses that it knows to be running the target service. These hosts are not necessarily vulnerable, but can be scanned without causing a connection failure which can help the worm to evade

some types of detectors. The number of new hosts (referred to as "neighbors") the worm discovers is dependent on its neighbor detection algorithm. We use three implementations of the topo worm with differing neighbor counts. The topo100 worm starts with 100 neighbors, the topo1000 worm starts with 1000 neighbors, and the topoall worm starts with an unlimited supply of neighbors. After scanning its known neighbors, the topo worm must either stop scanning or switch algorithms. In our implementation it reverts to random scanning after exhausting its neighbor list. Note that the neighbors discovered by the topo worm are randomly located, so could appear both inside and outside the protected network. Also, they will be running the target service but are not guaranteed to be vulnerable.

GLOWS accurately models the connection-level interaction between two hosts during an infection attempt, right down to setting appropriate TCP flags. Scanning attempts to non existent hosts result in SYN/RST exchanges as one would expect to see in the real world, and the worm client can disconnect at any point leaving the connection in an arbitrary state. The payload and target port number are also configurable.

We run GLOWS 16 times for each scan type in each environment. Each run of GLOWS uses a different seed for the random number generator. This gives us a variety of traces for each scenario to eliminate situations where a bad random seed gives us peculiar behavior.

We run the worm simulation outside of the evaluation engine for simple worms because it improves overall performance as we do not need to replicate the worm simulation for different detector types. However, some scenarios require implementing the worm inside the evaluation engine.

4.4. Evaluation

The core component of our system runs the actual detectors to measure their performance. It is by far the most complex component and contains the most code. The evaluation engine is written in Java and is comprised of approximately 17,000 lines of code.

The basic task of the evaluation engine is to read the network connections for a given network trace and simulated worm, and to run the selected detector against those connections, recording the output of the worm detector. Because the evaluation engine knows which connections are worm connections and which are legitimate, we are able to measure the detectors ability to detect worm traffic.

Before a detector can be run against worm traffic though, it must establish the thresholds that define anomalous behavior by training against benign, or legitimate traffic. Furthermore, we must establish each detector's false positive rate by running it against a period of benign traffic that contains no worm traffic. Finally, we must ensure that the false positives triggered when the worm is not present are not considered successful detection in future runs using the same benign traffic that do include worm traffic.

4.5. Experiment Coordination and Results Processing

The final portion of the framework is a set of scripts that pull it all together. The script collates the data from the different experiments and automatically generates specific graphs and summaries.

CHAPTER V

A COMPARISON OF BEHAVIOR-BASED WORM DETECTORS

The comparison described in this chapter was performed in conjunction with Dr. Jun Li and was published in the 2010 edition of the *Proceedings of the Recent Advances in Intrusion Detection (RAID) Symposium*. I was the primary contributor for both the underlying research and the writing, but Dr. Li helped to guide the research and edit the text.

5.1. The Selected Worm Detectors

Having selected detectors for our comparison work, we now describe them each in more detail. We present only a brief a summary of each work, please refer to the original publications for more detail. We use the same parameter choices that were presented in the original publication of each work. Note we used existing acronyms for each work where available.

The TRW detector was published by Schechter et al. in 2004 [35]. TRW identifies a host as worm infected if connection attempts to new destinations result in many connection failures. TRW is based on the idea that a worm-infected host that is scanning the network randomly will have a higher connection failure rate than a host engaged in legitimate operations. Even with the IPv4 address space getting closer to complete allocation, the majority of addresses will not respond to a connection attempt on any given port. Randomly targeted connections (as in

worm scanning) will likely fail. A TCP connection is only deemed successful upon completion of the three-way handshake. A UDP connections is considered to be failed when there is not a return UDP packet with the same source or destination port as the originating UDP packet within a timeout period. A higher connection failure rate of hosts can be efficiently observed by employing *sequential hypothesis testing*. Two hypotheses are established, one that the host is engaged in legitimate behavior (and therefore has a high connection success rate), the other that the host is infected with a worm (and therefore has a low connection success rate). Each connection to a new destination is observed as either a failure or success, and likelihood of each hypothesis is calculated. When the likelihood exceeds a given threshold for one hypothesis then it is considered to be true.

The TRW algorithm has the following parameters:

$\theta - multiple$: θ is the probability with which legitimate hosts make connections to new destinations successfully. The $\theta - multiple$ is a value between 0 and 1 used to calculate the probability with which worm infected hosts make connections to new destinations successfully. We use the $\theta - multiple$ chosen in [85], 0.2.

α : The false positive probability. A user-configured parameter between 0 and 1 that provides a bound on the false positive rate (if all the assumptions about event independence hold true). We start with the α from the original publication of 0.00001 and then modify it to fix our false positive level.

β : The detection probability. Similar to *alpha*, this is a user-configured parameter that provides a bound on the detection rate. We use β from the original publication, 0.99.

History size : Only connections to new destinations are considered the TRW algorithm, but TRW cannot remember all the visited destinations. The final parameter is the history size for establishing whether a destination is a new destination or a previously visited destination. We use the history size from the original publication, 30 minutes.

The destination-source correlation detector (DSC) was published in 2004 by Gu et al. [37]. It detects a worm infection by correlating an incoming connection on a given port with subsequent outgoing infections on that port. If the outgoing connection rate exceeds a threshold established during training, the alarm is raised. A different threshold is maintained for each destination port.

The DSC detector uses the following parameters:

Window Size : After an incoming connection to a host, the DSC detector monitors outgoing connections from the host for a set amount of time. We use a 60-second window, which is established via the training algorithm outlined in the original publication.

$\sigma - multiple$: The threshold for each port is selected based on the mean of observed infection-like events plus a multiple of the standard deviation. Modifying the $\sigma - multiple$ changes the aggressiveness of the detector. We

modify this value for each environment to set a fixed false positive level after training.

The MRW detector was first published in 2006 [83]. It is based on the observation that scanning results in connections to many destinations, and during legitimate operations the growth curve of the number of distinct destinations over time is concave. As the time window increases, destination growth slows. This can be leveraged by monitoring over multiple time windows with different thresholds for each window. Each time a host contacts a new destination, its recent history of contacting distinct hosts is evaluated against a collection of window sizes and thresholds. If the number of new destinations for a host within a given window exceeds the threshold, the alarm is raised. The key element in the MRW detector is window-size and threshold selection. The system can choose shorter window sizes and low thresholds that will result in high false positives but lower detection latency, or less aggressive values that will result in lower false positives but higher detection latency. The false positives and detection latency are the security cost components of the system, and are modeled as a simple linear combination with a user-selected weighting factor. The system then seeks to minimize the overall cost in choosing window sizes and thresholds.

The MRW algorithm has the following parameters:

β : The weighting parameter in the cost model. Higher values of β make for a more conservative system with lower false positives but higher latency. We

use a β of 65536 as is selected in the original publication.

Min worm rate : The minimum worm rate to input to the window-size and threshold selection algorithm. We use the minimum worm scan rate that we evaluate the detectors with, 0.005 scans per second.

Max worm rate : The maximum worm rate to input to the window-size and threshold selection algorithm. We use the maximum worm scan rate used in our evaluation: 10 scans per second.

Worm rate step : The size of the incremental changes to the worm rate to select worm rates for the window-size and threshold selection algorithm. We use the value from the paper of 0.1.

The RBS detector was first published in 2007 [93] by Jung et al.. Similar to the MRW detector, RBS measures the rate of connections to new destinations. The work is based on the hypothesis that a worm-infected host contacts new destinations at a higher rate than a legitimate host does. RBS measures this rate by fitting the inter-arrival time of new destinations to a exponential distribution. Each time a new destination is contacted, the RBS evaluates the likelihood of two hypotheses: that the host is benign and is contacting destinations at a low rate, and that the host is worm infected and is therefore contacting destinations at a higher rate. When the likelihood of the worm infected hypothesis exceeds a threshold, the alarm is raised that the host is infected.

The RBS parameters are quite similar to the TRW parameters.

$\lambda - multiple$: λ is the rate at which benign hosts make connections to new destinations. The $\lambda - multiple$ is used to derive rate at which worm-infected hosts make connections to new destinations. We use the $\lambda - multiple$ chosen in the original publication, 10, except in the wireless environment where we were unable to set the false-positives to the desired level without adjusting the $\lambda - multiple$

α : The false positive probability. Identical to the value for the TRW detector.

β : The detection probability. Identical to the value for the TRW detector.

Training window size : The RBS system continuously trains itself, monitoring the hosts in the network for a period, then deriving the new λ value from the observed training. We use the training window size of 10 minutes from the original paper.

History size : Similar to the TRW detector, RBS maintains a history to determine whether a connection is to a new destination or not. We use the value from the publication: 30 minutes.

The TRWRBS detector was published alongside the RBS detector [93]. It combines the TRW and RBS detectors into a unified scheme, and observes both the connection failure rate and the first contact rate. It performs a sequential hypothesis testing on the combined likelihood ratio to detect worms.

The TRWRBS detector uses the following parameters:

$\theta - multiple$: Uses a value of 0.6, from the original publication.

$\lambda - multiple$: Uses a value of 5, from the original publication.

α : Same as the RBS detector.

β : Same as the RBS detector.

Training window size : Same as the RBS detector.

History size : Same as the RBS detector.

The Protocol Graph detector (PGD) was introduced by Collins and Reiter in 2007 [94]. It is targeted at detecting slowly propagating hit-list or topologically aware worms. PGD works by building protocol-specific graphs where each node in the graph is a host, and each edge represents a connection between two hosts over a specific protocol. Collins and Reiter made the observation that during legitimate operation over short time periods, the number of hosts in the graphs is normally distributed and the number of nodes in the largest connected component of each graph is also normally distributed. During a worm infection, however, abnormal graph and largest connected component sizes are observed, indicating the presence of the worm. PGD is capable of giving some indication of which host is infected, but as this work is still in development we terminate processing as soon as PGD identifies that a worm is present in the network.

The PGD detector uses the following parameters:

Window Size : The PGD detector monitors the network for a set amount of time before checking the graph sizes. We use the window size from the published work, 60 seconds.

$\sigma - multiple$: The node count and largest connected component size thresholds are calculated for each protocol by adding a multiple of the observed standard deviation to the observed mean. We modify the false positive rate to set it at a fixed value after training by modifying the $\sigma - multiple$.

5.2. Detector Performance Metrics

The goal of this study is to evaluate the selected detectors over a comprehensive parameter space to identify their strengths and weaknesses. We must first, however, determine which performance attributes we are most interested in capturing, and what metrics would be suitable for assessing them.

There are many aspects to consider in the the overall performance of a worm detector. One could look at it from its accuracy in identifying infected hosts or worm traffic, the speed with which it makes a determination of whether a host is infected, the runtime processor or memory requirements, the types of worm propagation that it can detect, the complexity of configuring or installing it, and so on. Once a particular performance attribute is identified, one then needs to determine what metric to use. This is not as simple as it sounds. Accuracy, for example, can be measured in a variety of related, but different, ways including:

false positive rate (claiming a worm is present when one isn't), false negative rate (failing to identify the presence of a worm), ROC curve (a combination of the above two illustrating how they interact), sensitivity (the portion of worm infected hosts that are identified as such), specificity (the portion of non-worm infected hosts that are identified as benign), or F-score (a weighted combination of sensitivity and specificity) to name just a few. Let us first discuss the performance attributes we are interested in capturing, then the metrics we will use to measure them.

The focus of this study is on the ability of the detectors to discover the presence of a worm in the network. We thus want to measure their accuracy: does a detector alert us when a worm is present—but not do so when there is no worm? Furthermore, we want to measure its ability to detect a broad range of worm scanning algorithms. Moreover, accurate detection is not helpful if it happens too far after the fact. We must obtain some notion of the speed of the detectors—does it find a worm quickly or does it allow the worm free action for a long time before raising the alarm.

There are some attributes that we are not as interested in. At this time we are ignoring runtime costs such as processing or memory requirements. These are dependent on implementation and optimization details, and can vary widely for a given detection algorithm (for example, see the hardware implementation of TRW by Weaver et al. [27]). It is beyond the scope of this work to attempt to determine how efficiently each of these algorithms could be implemented. Similarly, we do not

consider the complexity of installing or running the detector. This is not because installation complexity does not impact the potential adoption rate of a detector, but because it is somewhat orthogonal to the accuracy of the detector itself and could be addressed separately from the detection algorithm itself.

As shown in Table 5.1, we have identified four metrics as the most useful measures of the performance of a worm detector. We explain them below:

Our primary performance attribute is accuracy in detecting a worm. We use two metrics to characterize accuracy: the false negative rate and the false positive rate. We have selected these metrics because they are easy to understand and are more relevant to worm detection than other metrics. Specificity and sensitivity are more common metrics, but are designed around reporting the performance of a test that is applied once or infrequently. A worm detector is constantly testing the network to see if an infection is present making the specificity and sensitivity values difficult to understand. This is known as the "base rate fallacy" in intrusion detection, a test can show what appears to be excellent sensitivity or specificity but perform poorly in practice because of the frequency with which the test is applied. We address this problem by reporting not the per-test rate, but instead reporting the aggregate number of false positives and false negatives seen at the experiment level.

Our false negative metric works as follows. For each experiment we introduce a worm to the background legitimate traffic. The detector is limited to a time

TABLE 5.1. **Metrics**

F-	Percentage of experiments where worm traffic is present but not detected in time period τ
F+ by host	The number of false alarms raised during a time period τ, limited to at most one false alarm per host
F+ by time	Perce ntage of minutes during a time period τ where a false alarm is triggered for any host
Detection Latency	The number of outbound worm connections from an infected network prior to detecting the worm

period τ (typically an hour) to detect the worm after it becomes active. If in that time span an alarm is not raised, the experiment is scored as a false negative for the detector. The *false negative rate* (F-) is the percentage of experiments scored as false negatives. (We report F- for each different scanning rate of the worm.)

The flip side of false negatives is false positives: reporting legitimate traffic as a worm infection. This is a critical metric for worm detectors, because a detector that repeatedly raises a false alarm ("cries wolf") will quickly be ignored by network administrators. We measure false positives by running the detector against benign traffic with no injected worm activity. (Because we have inspected the traces for known worm activity, we consider every alarm raised by a worm detector a false alarm.) However, because worm detectors often repeat their worm infection tests—on every connection in some cases, the same set suspicious behavior may cause the alarm to be raised repeatedly, and these repetitive alarms should be coalesced into a single notification to the network administrators. But the exact mechanism and scope of alarm coalescing will be specific to the needs and resources

of the network administrators at each site. As a result, we present two forms of false positive rate. We present the number of hosts identified as infected (coalescing alarms by network address) as the *false positive rate by host* (F+ by host). We also define *false positive rate by time* (F+ by time), which is the fraction of minutes of the trace where an alarm is raised on at least one host; note the alarm duration is only until the end of the current minute as we coalesce alarms into 1 minute bins. The combination of these two metrics give a better view of the overall false positive performance of the detector than either does individually.

The next major performance attribute to consider is the speed with which a worm is detected. The faster detection occurs, the less damage the worm can do because the fewer additional hosts it will have the opportunity to infect. We measure *detection latency* as the number of outbound worm connections initiated by all infected hosts in the protected network prior to detection of any internal infection. (Scans that do not leave the network do not inflict damage on the Internet as a whole and are not included in this count.) Alternative approaches such as using clock time or infected host count are less accurate and less descriptive than our metric.

An alternative approach to measuring latency would be to measure it in terms of clock time. This turns out to be a poor way of measuring because the slower a worm scans, the more clock time it will take for detection without changing the amount of damage the worm is doing. Another possible metric would be the

number of hosts within the protected network that are already infected prior to detection. However, our experiments show that when a monitored network is not very large, the number of infected internal hosts is almost always one at detection time, making this metric uninteresting.

5.3. Experiment Design

We run the detectors against legitimate traffic to measure false positives, then against legitimate traffic plus known worm traffic to measure false negatives and detection latency. We developed a custom testing framework and implemented each detector in our framework based on the detector's published specifications. Our framework can run against online, real-time traffic on the DETER testbed [97], as well as run in an offline (not real-time) mode. We use legitimate traffic from a variety of sources and generate known worm traffic by simulating a worm with our GLOWS [96] simulator. We vary the following parameters as we evaluate each worm detector: the environment it is run in (meaning the network configuration and legitimate traffic), the worm scanning method, and the worm scanning rate.

We evaluate each detector against network traces containing legitimate traffic as well as ones containing both legitimate traffic and known worm activity. We use legitimate traffic from a variety of sources. We split the legitimate traffic into training and experiment segments. After training a detector against the training segment, we evaluate it against the experiment segment. We use our GLOWS [96] worm simulator to generate worm traffic for each experiment. GLOWS takes as

input information about the legitimate trace, including the network configuration, host addresses, as well as host activity levels. It then simulates a worm for a given set of parameters and produces a worm trace, which is then merged with the legitimate trace to provide a trace with both legitimate and known worm traffic.

Evaluation Environment and Background Traffic

Worm detectors must be evaluated in the context of a subnet to be protected and against the legitimate background traffic that occurs in that subnet. This has traditionally been one of the difficulties in comparing published results of different detectors, they were evaluated in different scenarios with different traffic and were therefore not directly comparable. Real network traffic contains sensitive information and is not generally released to the public, making it difficult to acquire good traces to evaluate against. This difficulty has led researchers to each publish on privately acquired traces such that experiments cannot be repeated. For our experiments, we define an *environment* as the network address space to be monitored, the IP addresses of the active hosts inside that address space, and the IP network traffic into and out of that address space during two time periods. We use the first time period for training and the second to run experiments against. To make the environments comparable to each other and to enable us to ensure that they do not contain worm traffic, we select a /22 subnet from the original recorded traces to use as the protected address space in our environment. Every environment is thus a /22 network with between 100 and 200 active hosts. We use four distinct

environments in our evaluation.

Summary statistics of the traces can be seen in Table 5.2. These traces provide us legitimate background traffic from different time periods and different settings. They have a heavy academic leaning, which may influence our results, but they represent as wide a range of background traffic as we could acquire.

The *enterprise* environment is built from a trace collected at LBNL [98] (a member of the national laboratory system managed by the University of California) in January of 2005. The trace was captured in one-hour segments at individual router ports, and each segment sees only a subset of the overall traffic. Heavy scanners were removed from the trace before it was released. It has 139 active hosts and the training and experiment segments each contain roughly 25,000 connections.

The *campus* environment is built from a trace that was collected in 2001 at the border of Auckland University [99]. It contains over a month of traffic from the entire university with two /16 networks and several /24 networks. There are over 6,000 active hosts generating substantial traffic on a variety of protocols. The trace was anonymized using a non-prefix preserving anonymization scheme, so we cannot entirely accurately reconstruct the internal structure of their network. Instead, we randomly select 200 hosts and construct an environment using traffic to and from those hosts. Each segment of the trace in our campus environment contains approximately 25,000 connections.

The *wireless* and *department* environments are built from traces collected

at the University of Massachusetts in 2006 [100]. The department environment is built from a trace capturing all traffic to and from the wired computers in the CS department. It has 92 active hosts and approximately 30,000 connections in each segment. The wireless environment comes from a trace capturing all wireless network traffic from the university. It has 313 active hosts and approximately 120,000 connections in each segment.

We cannot establish ground truth for any one of these traces as the IP addresses have been anonymized and there is no connection payload recorded. This leaves us in an unenviable position of trying to evaluate detectors against traces with unknown quantities of worm activity present. The traces are also too large to carefully vet each host's behavior for normalcy. To mitigate these problems, we randomly select a much smaller subnet from the trace and concentrate our analysis on that. For each trace we selected a /22 network from within the trace containing between 100 and 200 active hosts. This small network size represents a realistic workgroup or department size and allows us to establish the legitimacy of the traffic with a higher degree of confidence.

Worm Parameters

Several key parameters of a worm may impact the effectiveness of worm detectors. Worms can use a variety of methods to find new targets to infect. We start our investigation by examining naive worms that use straight-forward scanning strategies. We examine the following scanning strategies at a variety

TABLE 5.2. **Trace statistics**: for each trace we list the number of active hosts inside the network, the total number of connections in the experiment segment, the percent of those connections that are initiated inside the network with a destination outside the network, and the percent of the total number of connections that use the TCP protocol.

Name	**Active Hosts**	**Connection Count**	**Outgoing Portion**	**TCP Portion**
Enterprise	139	25042	76.3%	50.6%
Campus	117	22935	66.2%	86.4%
Department	92	29634	53.0%	48.0%
Wireless	313	120032	72.3%	59.8%

of scanning rates: random scan, sequential scan, permutation scan, and local-preference scan. In practice, the evaluated detectors perform identically against the random scan, sequential scan, and permutation scanning worms, so we show on the results for the random and local-preference scanning worms. In this experiment the worms are naive, and make no effort to evade detection.

Our GLOWS simulator takes an environment as input and simulates a worm as if it were attacking the network defined by that environment. The simulation starts with a single inbound worm connection that infects one host in the protected network. We run the simulator once for each permutation of worm parameters. The scanning mechanisms are defined as follows.

See Section 4.3 for more detail about the scanning strategies of the worms used in this study.

In addition to scanning mechanism the worm uses, the rate at which it initiates connections is important. The faster a worm scans, the more visible it

is to worm detectors. To illustrate this point, consider a worm scanning at 1000 connections per second versus one scanning at only one connection per day. The worm with the higher scanning rate is a more significant disruption to "normal" traffic than the worm with the slow scanning rate We run experiments for a variety of worm scanning rates ranging from 10 connections per second down to one connection every 200 seconds. This allows us to determine which detectors are more sensitive and able to detect the slower moving worms.

The final variable parameter is the activity profile of the host that is first infected with a worm. Some hosts within the network have substantially more network traffic than others. We split the hosts into two categories: *busy* hosts are the 16 busiest hosts in the network, *idle* hosts are the 16 least busy hosts in the network, and *random* hosts are the 16 randomly selected hosts in the network. This allows us to examine how the legitimate traffic generated by a host impacts a detector's ability to discern the worm activity.

Experiment Procedure

Measuring detector performance is a multi-step procedure. For each environment, every detector must (1) establish thresholds via training, (2) be evaluated against the legitimate traffic in the environment to measure false positives, (3) adjust their parameters to fix false positives at a specific level, and (4) be evaluated against legitimate traffic combined with worm traffic to measure false negatives and detection latency. Let us now discuss each of these steps in more

detail.

These detectors are anomaly detectors, and they look for traffic that diverges from normal. To do this, they must first measure what normal is. The TRW, MRW, DSC, and PGD detectors are run against the training segment of the trace using the training method outlined in their publication to perform this operation. The RBS and TRWRBS detectors perform on-the-fly training as they are run against the experiment segment of the trace.

After the thresholds are established from the training segment of the trace, each detector is run against the experiment portion of the trace to measure false positives. We measure F+ using the thresholds obtained from training and the default detector parameters outlined in the original publication of each work, presenting those results in Section 5.4.

Note that each detector can be tuned to favor producing either more F+ or more F-. After reporting F+ using the default detector parameters as published, in order to provide a fair comparison of the false negative rate of the detectors, we modify each detector's parameters such that they all produce the same number of false positives in each environment. We chose to peg each detector at a rate of two false positive alarms during the experiment period. Two false positives is a high rate for the one-hour time period evaluated, but was chosen as an achievable value for all detectors requiring the minimum amount of parameter modifications.

After measuring F+ and adjusting the detectors to match their F+ levels,

we then measure the performance of the detectors against worm traffic. For each detector in each environment, we run 16 experiments for every permutation of the worm parameters. A single experiment consists of running the detector for 10 minutes of the experiment trace to warm up the connection histories, then injecting the simulated worm traffic into the trace, and running until either an hour has elapsed or the worm is detected. Each of the 16 experiments that we run for a given set of worm parameters has a different host in the protected network being infected first and uses a different random seed. The percentage of experiments where the worm is not detected is the false negative rate, and the mean number of worm connections that have left the network at detection time is the detection latency.

5.4. Results

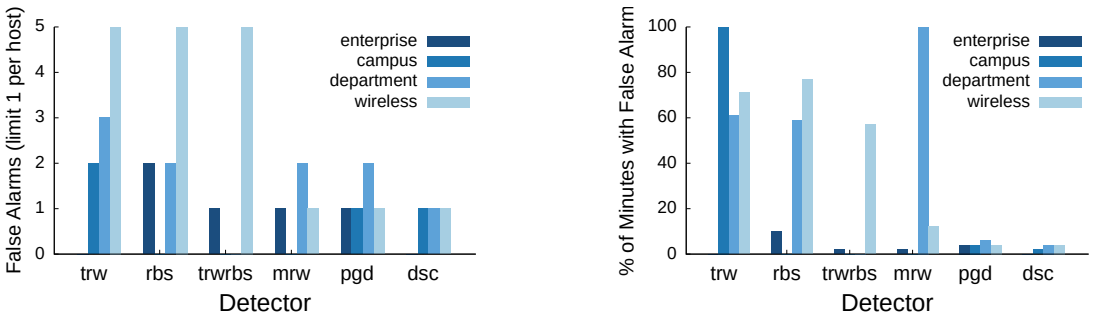
We now measure the performance of the worm detectors in a variety of worm scenarios. We first look at the false positives, then introduce worm traffic to measure false negative rates and detection latency, first for random scanning worms, then for local-preference scanning worms. We then look at the impact of the activity profile of the first infected host as well as the worm attack port.

False Positives Against Legitimate Traffic

Figure 5.1a shows the results for each detector using default parameters from its original publication. Raising an alarm for a host could either (a) indicate

that the host is considered permanently infected, or (b) indicate that the host is behaving anomalously *now* (for some definition of now). Figure 5.1a shows F+ results using strategy (a) (with PGD limited to one alarm per 1-minute window because it does not identify the infected host). Figure 5.1b shows F+ results using strategy (b) and with an alarm duration of one minute. Strategy (a) is probably more representative of how detectors would be deployed in practice, but it is illustrative to show that without such a limitation, in some environments RBS and TRWRBS would be in an alarm state more than 50% of the time and TRW and MRW would be in an alarm state 100% of the time.

These results also demonstrate the impact that environment has on the detector performance. TRWRBS has five F+ in the wireless environment but none in the campus or department environments. MRW is in an alarm state 100% of the time in the department environment but not at all in the campus environment. An evaluation using only a single environment could produce grossly inaccurate results.



(a) By Host

(b) By Time

FIGURE 5.1. **False positives against legitimate traffic:** when running with default parameters against the experiment segment of the traces with no worm traffic injected

The wireless environment showed the most F+ activity with the default parameter choices. This appears to stem from several hosts playing network games such as Counter-Strike (UDP connections on ports in the 27010-27050 range) and NeverWinter Nights (TCP connections on port 5121) as well as from hosts using BitTorrent (33 hosts active on ports in the 6881-6999 range). This environment represents the most residential/recreational usage patterns and indicates that this sort of traffic is less amenable to behavior-based worm detection than the less variable traffic of the enterprise environment. This represents the first findings we are aware of that validate a common hypothesis: current behavior-based anomaly detectors are not optimized for residential style network traffic and may not show satisfactory performance in such an environment.

The above results were with the default parameters for each detector. However, detectors can be adjusted to be more aggressive at identifying hosts as infected, causing more false positive results as a side effect. To ensure a balanced comparison between the detectors, after measure the false positive levels with default parameters we modify the parameters to result in 2 false positives during each one-hour experiment trace. This certainly represents a higher than desired level of false positives for such small networks and a short time period, but it is achievable by all detectors with the smallest changes to the parameters.

Table 5.3 shows the default parameters and the values they were changed to for each environment to achieve the desired level of false positives. In some cases

the parameter changes make the detector more aggressive and increase the number of false positives, while in other cases they make the detector less aggressive to reduce the number of false positives. For the TRWRBS and MRW detectors in the campus environment we were unable to achieve exactly two false positives. In these cases we set the parameters at the highest levels that generated one false positive. The DSC detector never generated false positives in the enterprise environment because there were no outgoing connections triggered by incoming connections. In this case we set the $\sigma - mult$ to zero to make the detector as aggressive as possible.

When analyzing false positives there is always the question as to whether there is actually a worm present in the trace. We limited the network size in each environment to enable us to review connection activity by hand. We found no obvious worm traffic, but because we have packets headers only there is no way to sanitize the traces completely. We examined the false positives and found that the detectors frequently mis-identified different hosts as infected. MRW and RBS had F+ for the same host 4 times across all the environments. DSC and RBS also had F+ for the same host 4 times, but only 1 host was a F+ for all three detectors. In no other cases did two detectors show F+ for the same host on more than twice. This indicates that the F+ are not dominated by a few very active hosts, but are generally different for the different detectors. This reinforces our belief that these are truly false positives, and not detection of unknown worm activity.

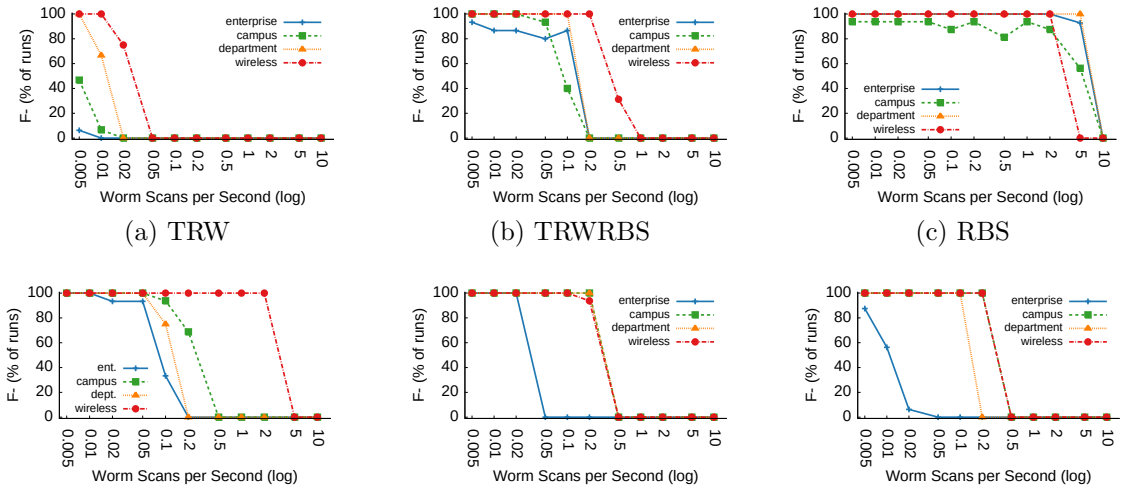
TABLE 5.3. **Parameter choices for the detectors.** The default parameters are what is suggested by the original publication of each detector. The parameters for each environment are what was chosen to achieve a false positive rate of two falsely identified hosts per hour during the experiment.

Detector	**Parameter**	**Default**	**Enterprise**	**Campus**	**Dept.**	**Wireless**
TRW	α	10^{-6}	10^{-3}	10^{-6}	10^{-13}	10^{-25}
RBS	α	10^{-6}	10^{-3}	10^{-9}	10^{-6}	10^{-6}
	$\lambda - mult$	10	10	10	10	1.5
TRWRBS	α	10^{-6}	10^{-4}	$3.9e^{-3}$	10^{-3}	10^{-18}
MRW	β	65536	16	51	65536	750
PGD	$\sigma - mult$	3.5	2.3	2.8	3.5	2.3
DSC	$\sigma - mult$	3	0	2.35	2.76	1.87

Detector Performance Against Random Worm

In this section we report false negative and latency results against random scanning worms. Figure 5.2 shows that TRW is the most consistently effective detector across the environments, discovering all instances of the worm down to 0.05 scans per second and catching the majority of the slower scans in the enterprise and campus environments. RBS is the least effective, only able to consistently detect the worm scan rates greater than five scans per second. TRWRBS blends the two detectors with results right in the middle. The DSC and PGD detectors are an order of magnitude more effective in the enterprise environment than in the other environments due to the lower activity levels (and hence lower thresholds) in the enterprise environment. The MRW detector provides middle of the road performance except against in the wireless environment where it is unable to detect the worm at speeds slower than five scans per second.

Figure 5.3 shows the average number of connections each infected network

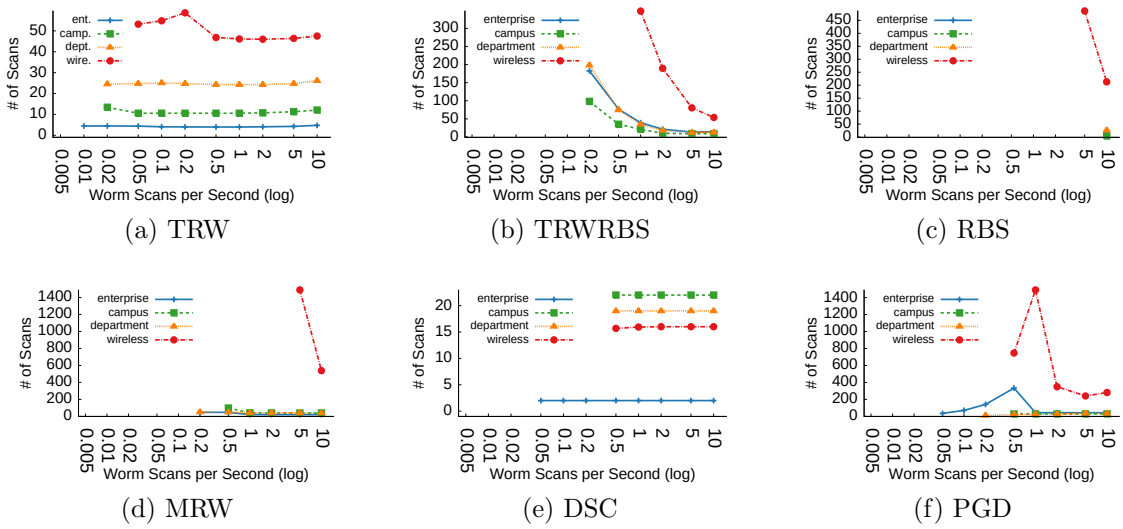


(a) TRW (b) TRWRBS (c) RBS

(d) MRW (e) DSC (f) PGD

FIGURE 5.2. **F- against random worm:** percent of experiments where the worm was **not** detected (lower is better performance) with a *random* scanning worm infecting randomly selected hosts and targeting port 80. For each environment and scanning rate we conducted 16 individual experiments using different first infected hosts and different random seeds. In each case the experiment was run until the worm was detected or one hour elapsed without detection.

was able to make before detection. Note that the scale is not consistent across the graphs. Recall that the detectors are based at the network gateway and observe only those connections that leave the network. The latency measure here is the number of scans the worm is able to make toward the outside network before it is detected. We only show the value for those scenarios where F- is zero in order to eliminate selection bias in the results. DSC is consistently the fastest detection mechanism, never allowing the worm to scan more than 23 times before detection. TRW again highlights the variation between environments, allowing roughly 50 worm scans in the wireless environment before detection, but only five scans in the enterprise environment. MRW and RBS allowed several hundred scans



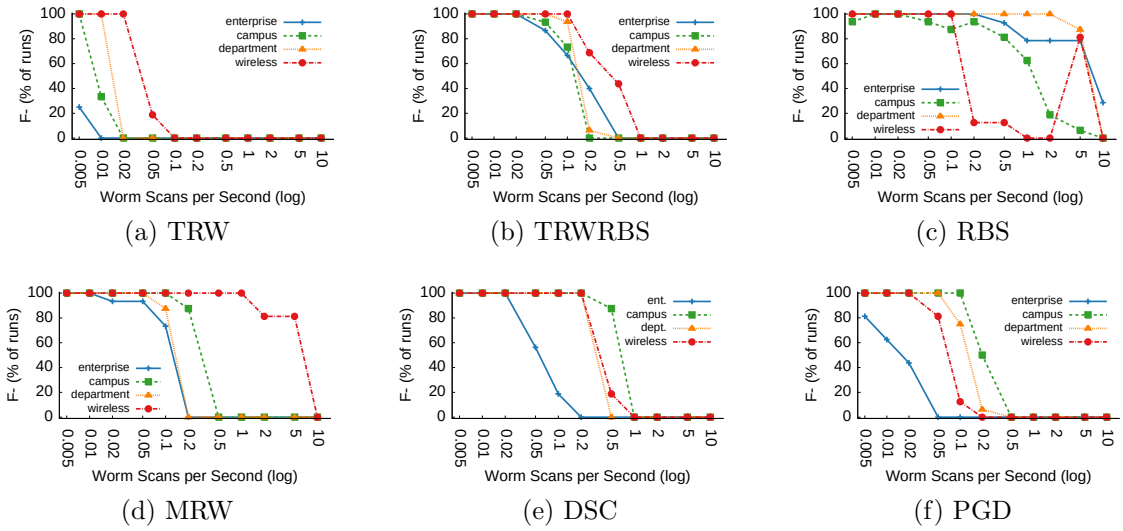
(a) TRW (b) TRWRBS (c) RBS

(d) MRW (e) DSC (f) PGD

FIGURE 5.3. **Latency against random worm:** from worm infection time to detection time for *random* scanning worm, measured as the number of worm connections leaving the protected network prior to detection. We report results only for those environments and scan rates where the worm was detected with 100% accuracy.

before detection in the wireless environment, but were much faster in the other environments. PGD showed the most variation, allowing over 1000 scans before detection in some scenarios in the wireless environment but detecting the worm in 30-40 connections in the other environments. TRWRBS showed increasing latency as the scan rate drops. This is due to the influence of the RBS algorithm that increases the destination threshold as the time window increases. The fast scanning worm is caught in a short window, but the slower scanning worms take a substantially longer time to hit the critical number of destinations.

Across the board, TRW shows the best detection performance against random scanning worms. This indicates that connection failures are a strong and highly identifiable signal. TRW also had consistent and low latencies, limiting the



(a) TRW (b) TRWRBS (c) RBS

(d) MRW (e) DSC (f) PGD

FIGURE 5.4. **F- against local-preference worm:** percent of experiments where the worm was **not** detected (lower is better performance) with a *local-preference* scanning worm targeting port 80.

damage a worm could do. Destination pattern based detection such as MRW and RBS typically requires greater numbers of connections for accurate identification and might be effective over longer time frames than the one-hour duration of our experiments. PGD performed adequately, but is designed to detect multiple infected internal hosts which did not happen with the random-scanning worm.

Detector Performance Against Local-Preference Worms

Having examined the baseline case using the random scanning worm, we now investigate performance against a more advanced foe: the local-preference scanning worm. The local-preference worm directs half its connections at the local network, meaning both that it is more likely to infect multiple hosts inside the protected network and that fewer connections per time period are visible to a gateway-based

detector. However, the scan is still random in nature, so shares the same general characteristics as the purely random scanning worm.

Figure 5.4 shows that for most scenarios, the detectors show a slight decrease in sensitivity. This is visible as a shift to the right in the false negative curves. The TRW detector was able to detect 100% of the random worms in the wireless environment at 0.05 scans per second, but is only able to detect 100% of the local-preference worms at 0.1 scans per second. TRWRBS, RBS, MRW, and DSC all show similar decreases in performance in some environments. The reason for this is simply the reduction of worm scans that are visible to the detector. The limit of a detector's ability to spot the worm—meaning the slowest worm that it can detect reliably—is at the point where it can just barely observe enough evidence to infer that a host is infected. If a worm scans more slowly or not all its scans cross the gateway (as in local-preference worms), the evidence visible to the detector may not be enough to make the determination that a worm is present.

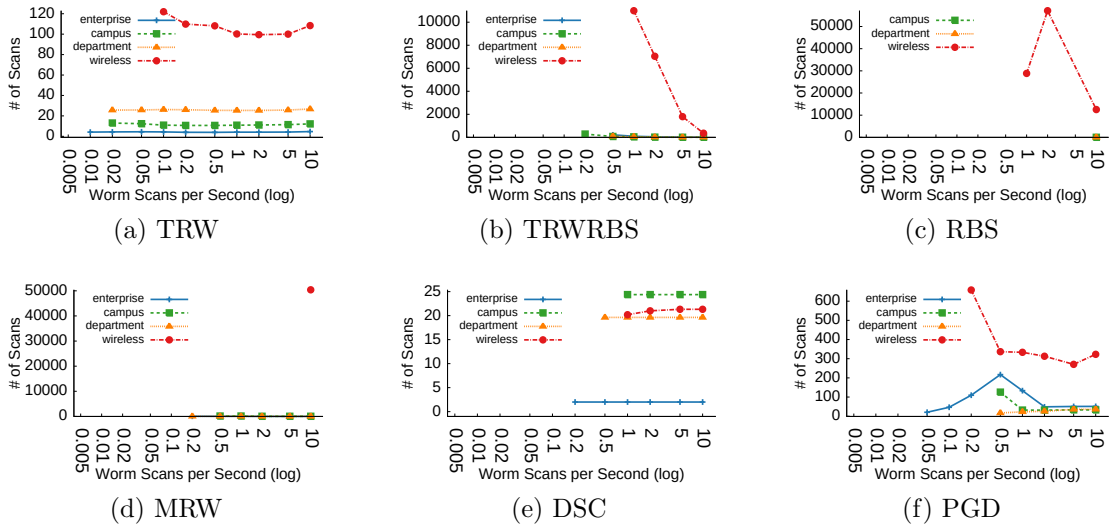
The one detector that shows a significantly different response is the PGD detector, showing *better* performance against the local-preference worm than it did against the random worm. The PGD detector measures the protocol graph of all hosts in the network, and the more infected hosts there are, the more scanning there will be using the protocol the worm targets. This leads to either more total nodes in the graph or a larger connected component, allowing the PGD detector to spot the local-preference worm in situations where it would not have detected a

random scanning worm.

The latency results are also impacted by the local-preference scanning strategy (Figure 5.5). The TRWRBS, RBS, DSC, and MRW detectors show worse detection latency in all environments for the local-preference worm as compared to the random worm. This is because the worm targets the local network so aggressively that in many scenarios it infects multiple hosts inside the network before it is detected. Recall that our latency metric measures the combined external scanning of *all* infected hosts in the network. The TRW detector, on the other hand, shows identical latency performance for all environments when comparing random and local-preference worms because it detects the worm before it infects *multiple* hosts (except in the wireless environment).

PGD behaves quite differently than the other detectors. It detects the local-preference worm more quickly than the random worm in the enterprise and campus environments, but slower in the department environment. And in the wireless environment the local-preference worm is detected more quickly at scanning rates of two scans per second or less, but the random worm is detected more quickly at rates above two scans per second.

The DSC detector is the fastest, allowing fewer than 25 outgoing worm connections in all scenarios where it was able to detect the worm 100% of the time. TRW is also quite fast, allowing fewer than 27 connections in all environment except for the wireless environment where it allows roughly 100. Note TRW also is



(a) TRW (b) TRWRBS (c) RBS

(d) MRW (e) DSC (f) PGD

FIGURE 5.5. **Latency against local-preference worm:** from worm infection time to detection time for *local-preference* scanning worm, measured as the number of worm connections leaving the protected network prior to detection.

the most sensitive detector, successfully detecting the worm at the lowest scanning rates in all environments.

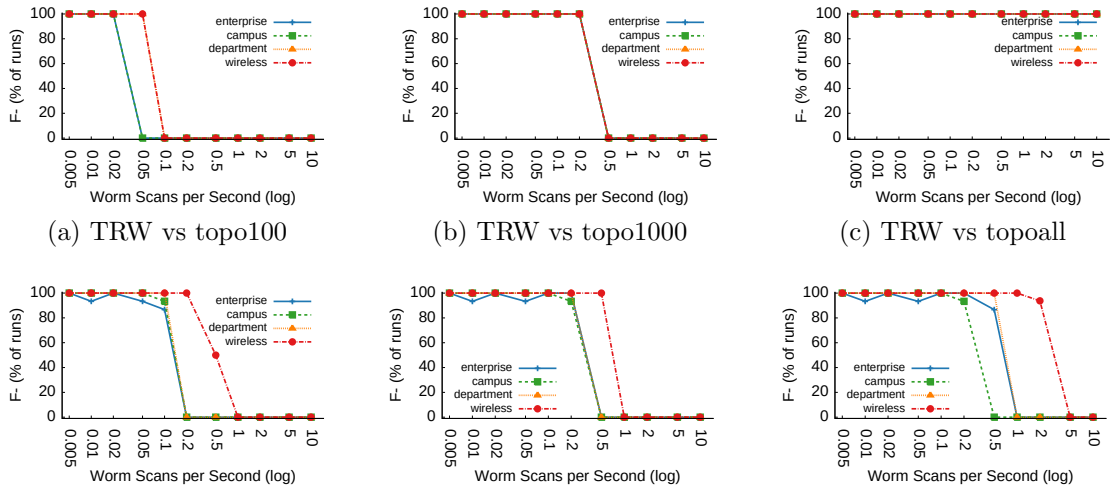
Evading TRW

Topo scanning changes the observed behavior of an infected host by reducing the number of connection failures that the detector can observe (See Section 4.3 for a complete description of the topo worm). The neighbors discovered by the topo worm are vulnerable at the same level as other hosts in the network but are guaranteed to be present, different from random scanning where a large number of scans go to addresses with no host present. The only detectors that are impacted by this strategy are those detectors that rely on observing connection failures: TRW and TRWRBS. The RBS, MRW, DSC, and PGD detectors show

identical performance against the topo worm and the random worm. The pattern of neighbors—whether they can be connected to or not—is random in both the random and topo worms and thus triggers those algorithms in the same way.

The TRW detector is unable to detect the topo worm during its topo scanning phase because of the lack of connection failures. It *only* detects the worm after it reverts to random scanning. In the topo100 scenario (Figure 5.6a), this occurs relatively quickly as it does not take long for the worm to exhaust its list of 100 neighbors. TRW is able to detect the worm at speeds as low as 0.01 scans per second in all environments. However, in the topo1000 scenario, the list of neighbors is not exhausted during the one-hour experiment for speeds below 0.5 scans per second and the TRW detector is unable to detect topo worms with slower scanning rates (Figure 5.6a). In the topoall scenario—where the topo worm never exhausts its list of neighbors—the TRW detector is *never* successful at detecting the worm (Figure 5.6c).

Not only is TRW's ability to detect the worm compromised, but even in scenarios where it does detect the worm it is much slower at it. Figure 5.7 shows the latency results for TRW against the topo worm. Because during the worm's topo phase none of its scans were detected, the latency results against the topo100 worm are approximately 100 scans worse than they were for TRW against the random scanning worm. Similar results can be seen for the topo1000 scenario, where TRW's detection latency is 1000 connections worse than it was for the



(a) TRW vs topo100 (b) TRW vs topo1000 (c) TRW vs topoall

(d) TRWRBS vs topo100 (e) TRWRBS vs topo1000 (f) TRWRBS vs topoall

FIGURE 5.6. **F- against topo worm:** percent of experiments where the worm was **not** detected (lower is better performance) by the TRW and TRWRBS detectors with a *topo* scanning worm targeting port 80. The topo100 worm uses 100 neighbors before reverting to random scanning, the topo1000 worm uses 1000 neighbors before reverting to random scanning, and the topoall worm never uses random scanning.

random scanning worm.

This shortcoming in TRW is one of the motivations for the TRWRBS detector. It uses connection failures in the detection algorithm, but it can also detect a worm even with no connection failures by checking the rate of connections to new destinations. The TRWRBS detector is able to detect the topo100 at rates above 1 scan per second in the wireless environment and above 0.2 scans per second in all other environments (Figure 5.6d). It does not perform quite as well as TRW in this scenario because TRW is able to leverage the connection failures so effectively. In the topo1000 scenario the detectors are effective at approximately the same worm scanning rate (Figure 5.6e); but if one looks at the latency, the

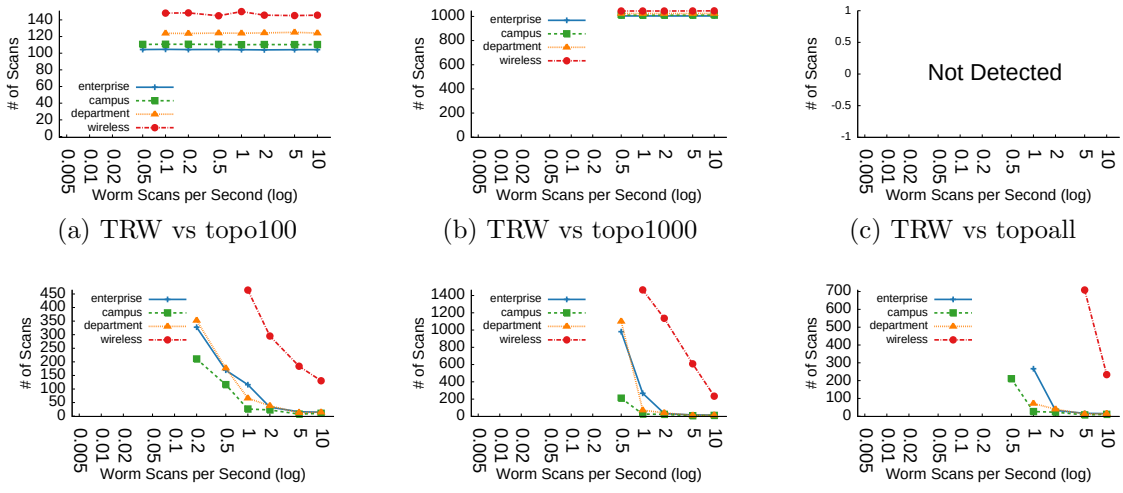


FIGURE 5.7. **Latency against topo worm:** from worm infection time to detection time for *topo* scanning worm, measured as the number of worm connections leaving the protected network prior to detection. The topo100 worm uses 100 neighbors before reverting to random scanning, the topo1000 worm uses 1000 neighbors before reverting to random scanning, and the topoall worm never uses random scanning.

TRWRBS detector is able to detect the worm more quickly at most scanning rates (Figure 5.7e). At worm scanning rates of 2 scans per second and higher, TRWRBS can detect the worm in under 30 connections in all the environments except for the wireless environment. This compares well against the TRW algorithm which requires over 1000 scans before detecting the topo1000 worm. The TRWRBS detector even detects the worm in the topoall scenario where the TRW detector could not.

This reliance on connection failures highlights a potential weakness of the TRW algorithm. If a worm can generate a big enough list of hosts running the target service that are likely to exist, it can make enough successful connections

to completely evade the TRW algorithm. The detectors based on destination distributions do not have this weakness.

Summary

Reviewing our findings, we can see that the TRW detector shows the best performance against naive worms. It can detect slower random and local-preference scanning worms than any of the other detectors in all the environments we tested. The PGD detector was capable of detecting all types of worms scanning at 0.5 scans per second or faster in all environments, but was relatively slow, frequently allowing several hundred scans prior to detection. The TRWRBS detector showed similar to the PGD detector. The RBS detector was only capable of detecting fast scanning worms. The MRW detector struggled to detect worms in the wireless environment and was incapable of detecting the local-preference worm in that environment. Finally, the DSC detector performed quite well in many respects, but is otherwise quite limited due to the requirement that an inbound infecting connection be observed in order for the detector to function. An initial infection that came via some other vector (removable media, direct download, etc.) would be undetectable by DSC.

The wireless environment was the most difficult for detectors to operate successfully in. In virtually all scenarios, detectors showed the worst sensitivity in the wireless environment, and detection latencies were typically an order of magnitude worse. The traffic in this environment is more focused around

entertainment type activities such as network gaming and peer-to-peer file sharing. These activities are prone to resembling worm scanning activity, making it more difficult for the detectors to differentiate between legitimate hosts and worm infected ones. For example, a peer-to-peer network client may receive a list of peers who were recently active and attempt to contact every host on the list. If the peer-to-peer network has a high churn rate and hosts on the peer list have left the network, this activity will result in many connection failures, just as if a worm were scanning for potential targets. Even in the face of this type of activity, however, the detectors were still typically able to detect true worm activity. As in the other environments, the TRW detector was able to detect slower worms than any other detectors. The PGD detector showed the next best performance.

Our results indicate that worms scanning at one connection per second or better are relatively easily detected in most environments. However, these are naive worms, in the next chapter we reconsider this detection performance in light of worms that attempt to evade detection.

CHAPTER VI

EVASIVE WORMS

The worms we looked at in the previous chapter are well known worms as published in "How to to 0wn the Internet" [8]. However, they use relatively naive scanning strategies that do not consider detection. In this chapter, we examine improving the worm scanning strategies in order to better evade detection. We first look at the basic principles behind evasion, then develop a series of evasive worms. We identify the metrics which best gauge their performance, and finally measure ability to propagate while evading detection.

The basic goal of the worms studied in this work is to propagate widely. A worm can propagate more successfully if it goes undetected. When a worm is easily detected in the network, countermeasures can be deployed against it. The worms in the previous chapter did not take this aspect of operation into consideration. They operated blindly without considering existing network traffic or what impact their additional traffic would have. It is almost laughable to consider a worm that would ignore such basic concepts, yet existing worms generally do not focus on network behavior based detection and so in fact do not take their network *behavioral* signature into account. Note that this not the case when examining the data signature of worms. Extensive research has been done in polymorphic worms that avoid sending a identifiable byte stream signature across the network.

A major contribution of this work is to examine the capability of network worms to modulate their behavioral signature and avoid behavior-based worm detectors. We consider the potential capabilities of such worms and examine their ability to evade actual detectors.

6.1. Evasive Worm Capabilities

The first step in evaluating evasive worms is to define a problem space. There are unlimited capabilities that a worm might have, from no knowledge of its operating environment (the typical naive worm), to knowing the location and supported services of every host on the targeted network, even to the point of having a group of coordinated worm infected hosts operating jointly (swarm activity [101]).

To simplify our evaluation, we first establish a small number of realistic capabilities that define our problem space. Future work may expand upon these capabilities, but these give a good starting point. We split capabilities into two categories: (1) knowledge of the infected host, and (2) knowledge of the target environment.

Knowledge of the infected host means knowledge of the network connections it initiates and receives. This is actually a fairly easily achievable goal, as any worm that achieves root access on its target host can monitor network activity. However, it is not a given, and as we will see in the evaluation results, the knowledge of local network traffic provides a non-negligible advantage to a worm.

The second capability is knowledge of the target environment. Specifically, this means knowing algorithm employed by the worm detector, and further knowing the parameters under which it operates. It is slightly harder to justify this knowledge, but we assume that a worm knows it via some out of band operation. It is conceivable that a detection algorithm could gain such prominence that it was the only one that is commonly deployed, which would give worms a good idea of what they are up against. Furthermore, it is probable that detectors would commonly be deployed with their default settings, making establishing the operating parameters feasible. We do not examine the matter of a how a worm would come by such information in this work, however. All evasive worms evaluated in this chapter have at the very least, knowledge of the detector employed against them.

In the remainder of the chapter we'll use the following terms to refer to the different capabilities of evasive worms as established above (see Table 6.1). A worm with no knowledge of the legitimate network traffic on an infected host is said to be *blind*, whereas if it can observe the traffic it is *perceptive*. A host that does not know the parameters of the detector deployed against it is described as *speculative* whereas one that knows the actual deployed parameters is said to be *informed*. We consider all permutations of these capabilities.

6.2. Evasion Tactics

The tactics for a worm that is trying to evade detection are simple. Know the

TABLE 6.1. **Evasive worm capabilities**

Capability	**Description**
blind	unable to observe legitimate on infected host or network
perceptive	able to observe all traffic on infected host or network
speculative	knows detector used against it, but does not know configuration
informed	knows detector used against it and all detector parameters

underlying details of the detector being used, then minimize behavior that would trigger detection.

The detectors we have evaluated employ three basic techniques for detecting worms. They look for an infection signature, an anomalous number of destinations visited, or an anomalous connection failure rate. Once a worm knows what detection technique is employed against it (remember, at this point we do not address how this knowledge is gained) it simply must adopt a suitable scanning technique to minimize this behavior. Let us now examine each of these behavior techniques individually.

Evading Infection Signature-based Detection

The DSC detector employs an infection signature-based technique for detecting worms. It correlates incoming connections to a given service with bursts of outgoing connections on that service. Evading a detector such as this is relatively straight-forward. The worm simply adds a delay between infection and the beginning of scanning. The DSC detector relies on a fairly short window between infection and subsequent scanning activity. If the worm waits longer

than this window, it will not be detected. It must, however, watch for additional incoming connections to this service that would then cause its scanning to be considered malicious. For this reason, the *perceptive* worm should be expected to perform better than the *blind* worm, because it can watch for incoming legitimate connections and pause its scanning when it observes them so as to avoid detection. An *informed* worm will be marginally more successful than a *speculative* worm because it will know precisely how long it must wait after infection before scanning.

Evading Destination Counting Detection

The MRW and PGD, and RBS, and TRWRBS detectors all employ some form of destination counting mechanism. This detection mechanism also happens to be the hardest to evade, because a worm **cannot** avoid contacting destinations, as that is how a worm must propagate. A worm can, however, moderate its scanning behavior. A *perceptive* worm can avoid scanning any time legitimate scans are made so as to avoid the additive impact of the legitimate and worm scanning connections. An *informed* worm will know the precise scanning rate that will trigger detection. Note that this information by itself will not necessarily allow the worm to evade detection entirely. This is because the connections are always measured over some time frame, and the worm cannot predict the future behavior of the host.

Consider a simplified example where a host is considered infected if it connects to more than 10 different destinations within a 10 second period. The

worm cannot simply scan at one connection per second and expect to evade the detector, because in any 10 second period where there was a legitimate connection, the alarm would sound (the 1 legitimate destinations plus the 10 worm destinations exceeding the threshold of 10). Instead, the worm must operate at some fraction of the threshold that leaves sufficient room to allow for legitimate traffic without exceeding the threshold. We will examine this more in Section 6.5.

Evading Connection Failure Detection

The TRW and TRWRBS detectors employ a connection failure counting mechanism. A naive worm that is scanning for hosts to infect will likely encounter an increased connection failure rate that can be used to detect the scanning. Both of these detectors are based on the connection failure *rate* (to new destinations only, not counting repeated connections to a single destination) rather than the total number of connection failures.

There are several methods that a worm may employ to evade detection by this type of detector. Because the failure rate is used rather than an absolute number of connection failures, if the worm has a list of known hosts that it can connect to it can use the successful connections to these known hosts to lower its failure rate. The TRW and TRWRBS detectors are not designed to keep their state forever, and "forget" about destinations after 30 minutes. This means that a list of known destinations can be re-used every 30 minutes, meaning that the list need not be excessively large. If a slow initial propagation rate is acceptable, a *perceptive*

evasive worm could build up its own list of known hosts by observing successful connections and remembering the destinations.

A second means of evading this type of detector is to use some other means of finding hosts to infect rather than simple scanning. Staniford et al. describe this type of worm as a *topological* worm. A worm that uses information found on the infected host to find additional hosts to infect. These hosts could come from lists of peers on the network, recently visited web and mail hosts, or a variety of other sources. Connections to these hosts would have a low connection failure rate, allowing the worm to avoid detection.

6.3. Methodology

We evaluate the evasive worms using the same techniques outlined in the previous chapter. Each worm detector is measured against a worm designed specifically to evade it. This presents the worst-case scenario for each detector to illustrate their potential weakness. We did not feel it necessary to evaluate each detector against all types of evasive worms, as that additional data would simply muddy the results that measure the robustness of each detector when specifically targeted for evasion.

We run each evasive worm in the same scenarios that were used in the previous section. Additionally, for each evasive worm we vary a parameter ζ between zero and one, controlling the aggressiveness of the scanning. A value of one, means that the worm will modulate the traffic it generates from its own

scanning to bring the detector exactly to its threshold. A value of zero would cause the worm to modulate its traffic so that it presented no signature to the worm (if possible). A value of zero for ζ of course, might mean that the worm would not scan at all. We use the term *load factor* because this worm traffic is additional load against the detector threshold beyond whatever legitimate traffic originates from the host.

Let us use an example to help illustrate the use of this parameter: imagine a simple rate based detector that would raise the alarm when a host made more than 20 connections per second. An evasive worm with a ζ of 1 would attempt to generate precisely 20 connections per second. The maximum possible without raising the alarm. This same evasive worm with a ζ of 0.5 would attempt to generate 10 connections per second, using half of the threshold for worm connections. This parameter is valuable because even a perceptive worm cannot predict future traffic. A worm that is tries to match the worm threshold exactly may frequently exceed the threshold and be detected.

We run each evasive worm once in each environment for each of 16 different randomly selected first infected hosts, for 10 different values of ζ in the range [0.1 - 1.0].

6.4. Metrics

Use three metrics to evaluate the success of an evasive worm. As we vary the load factor, we measure the worms ability to evade detection: its *evasion rate*.

This represents the percent of experiments where the evasive worm is not detected during the one hour run.

The second metric is the *effective scanning rate*. This is the average number of worm scans per second the evasive worm is able to make during the one hour experiment for a given environment and value of ζ.

The higher the value of ζ, the faster the evasive worm will scan, increasing its effective scan rate. This will also increase the chance of detection, decreasing the evasion rate. An evasive worm author's goal is to scan as quickly as possible while maintaining a fixed chance at detection. We call this allowable rate of detection ψ. By choosing a value for ψ we can then find the *maximum effective scanning rate* for an environment by finding the maximum load factor with an evasion rate of greater than $1 - \psi$. For this experiment we chose a value of .10 for ψ (10%). This means that an evasive worm will be detected in no more than one of the 16 experiments for a given load factor in a single environment. The maximum effective scanning rate is the ultimate determination of a worm detector's effectiveness. The lower the maximum effective scanning rate allowed, the more effective a detector is. This single metric is the best metric for comparing detectors, as it reveals the damage that a worm can cause without being detected.

The decision to base our metrics on detection on clock time rather than on the number of hosts infected may seem an odd one, but it is actually quite sensible. Our initial thought was to measure it based on the number of hosts infected before

detection, but this does not yield interesting results. An extremely slow worm (scanning once per week, for example) will operated undetected for an extremely long time and may eventually be able to infect all vulnerable hosts without triggering any detector (assuming a *very* patient attacker). However, this does not mean that the worm is particularly worthwhile. All the detectors we studied can be evaded by only scanning once per day. Limiting the detection window to one hour means that we're testing an evasive worm's ability to propagate as quickly as possible. It means that we truly test each detector's ability to slow down the worm and therefore to limit the overall propagation rate.

6.5. Evasive Worm Detection Results

In this section we present the results of our experiments against the six worm detectors evaluated in Chapter V. Each detector is measured against a custom evasive worm designed specifically to evade it. We use the same network traces, first infected hosts, and detector configuration parameters as were used in Chapter V. We limit an evasive worm's scanning rate to 10 scans per second. In some scenarios a worm could achieve a higher maximum scanning rate if we increased this value, but in those cases it is clear even with this capped rate that the detector is a poor performer, and increasing the rate would not add additional information. We first examine the DSC detector, which uses the *infection signature* as core heuristic. We then consider the MRW, RBS, and PGD detectors, which each use some form of *destination counting*. The TRW detector and it's *connection*

failure detection scheme follows, with the TRWRBS' hybrid *destination counting* and *connection failure* scheme last.

DSC

We first examine the DSC detector. It performed well against naive worms, with a very low detection latency and good sensitivity. The DSC detector, however, is trivially evaded. A worm must simply wait long enough after it infects a host before scanning to avoid any causality connection between the infection event and the scanning. Once the worm has waited long enough to avoid any causality connection, it may proceed to scan at arbitrarily high rates. The only way the worm will be detected is if there is an incoming connection which could trigger causality. Even in this case, a *perceptive* worm will detect the incoming connection and can stop scanning long enough to break causality. Listing 6.1 shows the core logic of the blind version of this evasive worm while Listing 6.2 shows the core logic of the perceptive version of this worm.

Figure 6.1 shows the achieved scanning rate by different types of evasive against the DSC detector as a function of load factor. For this worm, the load factor is inconsequential because of the trivial means of evading the worm. For most of the duration of the experiment the worm is able to scan at its maximum rate, yielding an effective scanning rate very near to 10.

Figure 6.2 shows that the worm is able to evade detection in all virtually all circumstances. The perceptive worms perform slightly better than the blind

LISTING 6.1. **DSC Blind Evasive Worm**

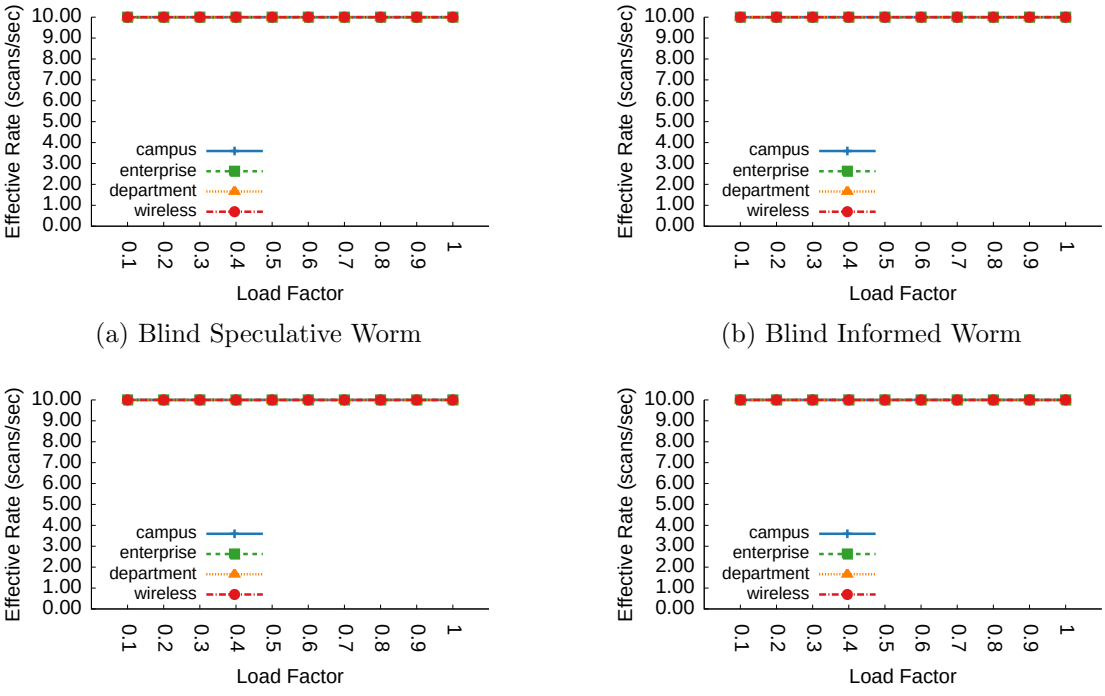
```
// timeout is time required between infection and scanning
function doScan() {
    if (now <= infectionTime + timeout) {
        // wait for timeout
        waitUntil(infectionTime + timeout);
    }
    // once timeout is passed, scan as fast as possible
    scanRandomTarget();
}
```

LISTING 6.2. **DSC Perceptive Evasive Worm**

```
// timeout is time required between infection and scanning
function doScan() {
    // check for recent infection activity
    lastInfectionTime = checkForInfectionActivity();
    while (now <= lastInfectionTime + timeout) {
        // wait the timeout period again
        waitUntil(lastInfectionTime + timeout);
        lastInfectionTime = checkForInfectionActivity();
    }
    scan();
}
```

worms because they recognize the incoming connections to one of the hosts in the department environment.

Figure 6.3 shows the *maximum effective rate* for an evasive worm with different capabilities against the DSC detector. These results show the importance of considering evasive worms when evaluating the quality of detectors. The DSC detector appears to be an excellent performer when evaluated against naive worms, but performs very poorly against a worm targeted to evade it.



(a) Blind Speculative Worm

(b) Blind Informed Worm

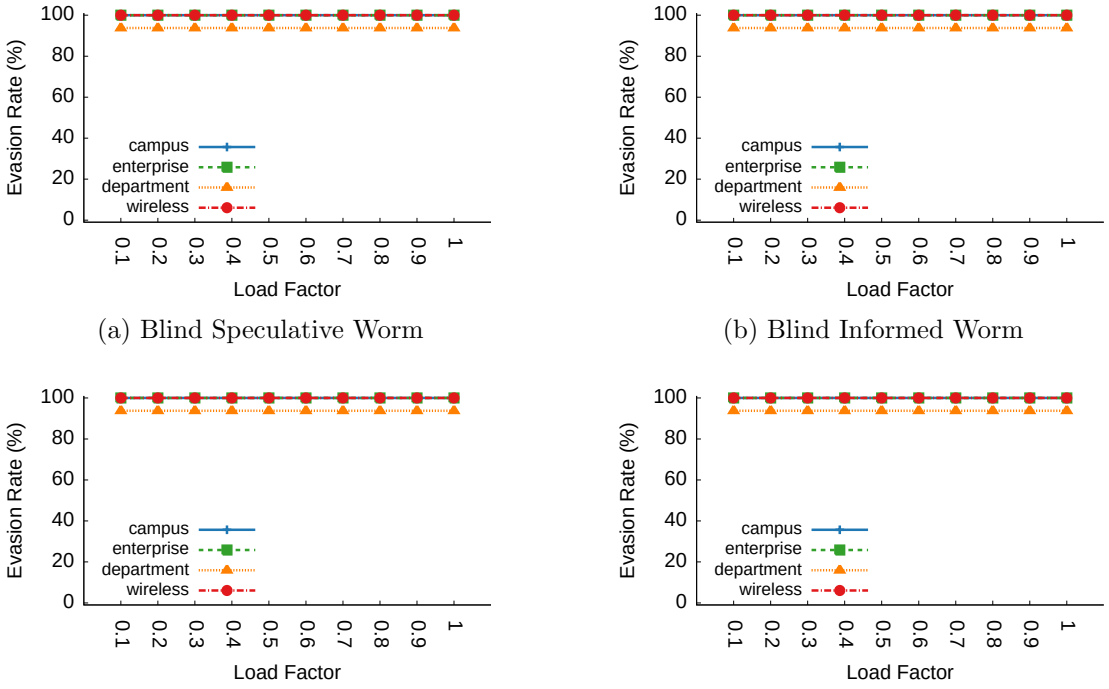
(c) Perceptive Speculative Worm

(d) Perceptive Informed Worm

FIGURE 6.1. **Effective scanning rate vs DSC** detector as a function of Load Factor

MRW

Unlike the DSC detector, the MRW detector puts significant constraints on the behavior a worm can exhibit and effectively limits its achievable scanning rate. The worm's evasion mechanism is to limit it's scanning rate to the fastest one that won't be detected by any of the detection windows. This algorithm may sound simplistic, but ultimately, it is the only one that can succeed against a detector of this type. The blind evasive worm assumes that no legitimate traffic originates from the infected host. The perceptive worm can observe legitimate traffic and so more accurately limit it's scanning rate. Listing 6.3 shows the core logic of the blind



(a) Blind Speculative Worm

(b) Blind Informed Worm

(c) Perceptive Speculative Worm

(d) Perceptive Informed Worm

FIGURE 6.2. **Evasion rate vs DSC** as a function of Load Factor

version of this evasive worm while Listing 6.4 shows the core logic of the perceptive version of this worm.

Figure 6.4a shows the effective scanning rate of the Blind Speculative MRW evasive worm. It achieves the same scanning rate across all environments because it uses the same conservative threshold values in all environments. It knows nothing about the environment it is deployed in so cannot make any adjustments to its scanning rate. Because it chooses a conservative threshold, it also shows fairly good evasion rates across the environments. (see Figure 6.5a). It shows the worst evasion rate in the enterprise environment, which is also the environment with the lowest thresholds. The Perceptive Speculative version of this worm shows slightly more

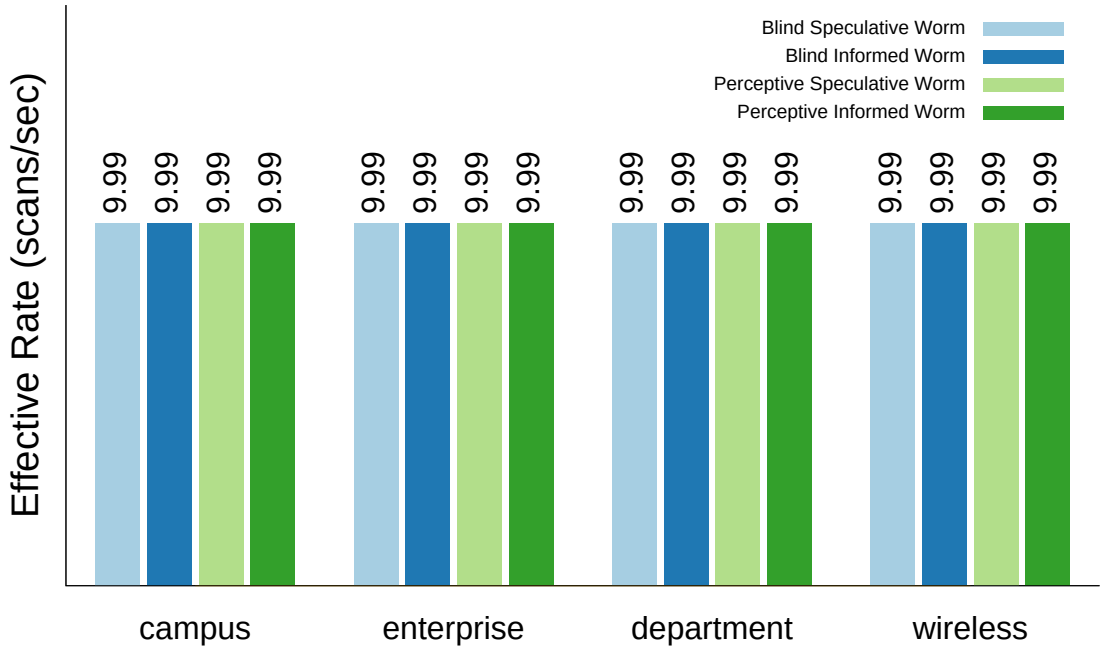


FIGURE 6.3. **Maximum effective rate vs DSC** as a function of Load Factor

LISTING 6.3. **MRW Blind Evasive Worm**

```
// interval is time between scans at maximum undetected scanning rate
function doScan() {
    wait(interval);
    scan();
}
```

variation in its effective rate (Figure 6.4c). This is due to the fact that the worm scales back its scanning as it sees legitimate traffic. This slightly helps the evasion rate of the worm (Figure 6.5c), but makes only a moderate improvement because the speculative thresholds are already conservative. It has the negative side effect of actually lowering the worm's effective rate.

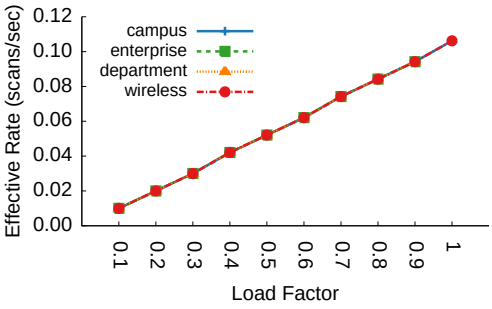
The Blind Informed version of this worm shows similar evasion rates (Figure 6.4b to the Blind Speculative version, with slightly worse evasion results

LISTING 6.4. **MRW Perceptive Evasive Worm**

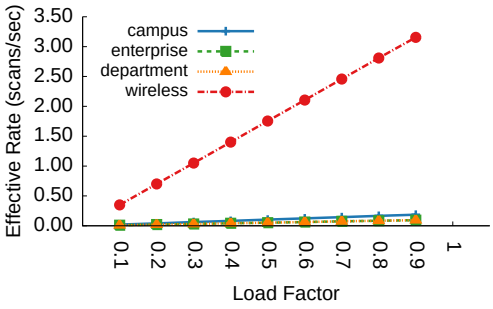
```
// interval is time between scans at maximum undetected scanning rate
function doScan() {
    wait(interval);
    updateScoreFromLegitimateActivity()
    if (! wouldRaiseAlarm()) {
        scan();
    }
}
```

as the load factor approaches 1.0. Knowing the threshold is most significant when looking at the effective rate of the worm. The Blind Informed worm was able to achieve an effective rate of greater than 3 scans per second in the wireless environment compared to roughly 0.1 for the Blind Speculative worm. The wireless environment has generally busier hosts and therefore higher thresholds, allowing an evasive worm to scan more quickly without being detected

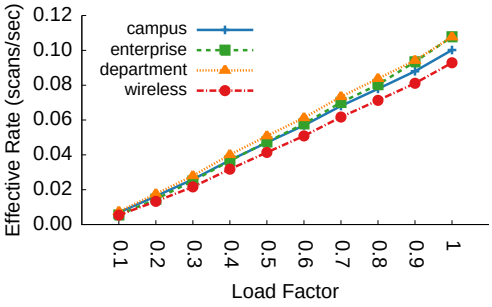
Figure 6.6 shows the maximum effective rates achieved by the different varieties of MRW evasive worm in the different environments. The most interesting feature of this graph is that with the exception of the wireless environment the different capabilities of the worm made little difference. In the campus environment the Informed worms saw roughly a 27% boost in scanning rate. In the enterprise and department environments, however the informed worms saw no advantage, but the perceptive worms showed 20% and 28% improvements in the respective environments. The wireless environment shows that in some cases, knowing more about the environment can make a significant advantage. The informed worms were able to scan 30x faster than the speculative variety.



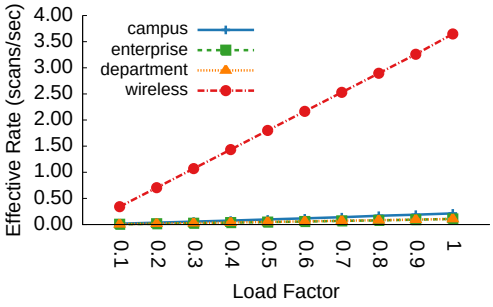
(a) Blind Speculative Worm



(b) Blind Informed Worm

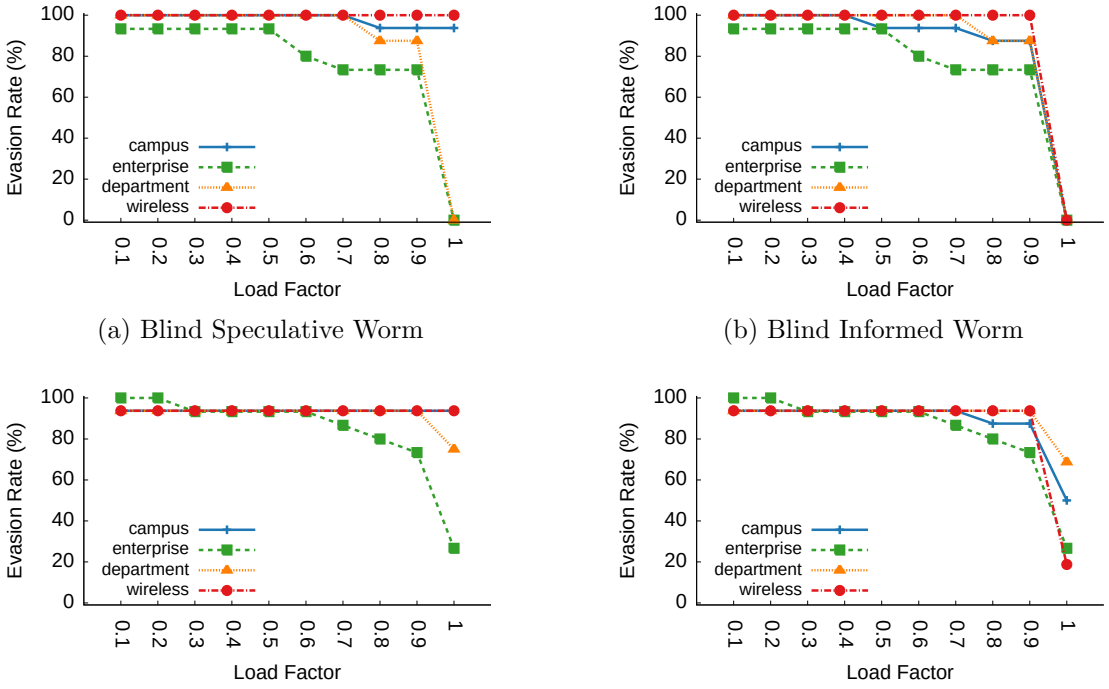


(c) Perceptive Speculative Worm



(d) Perceptive Informed Worm

FIGURE 6.4. **Effective scanning rate vs MRW** as a function of Load Factor



(a) Blind Speculative Worm

(b) Blind Informed Worm

(c) Perceptive Speculative Worm

(d) Perceptive Informed Worm

FIGURE 6.5. **Evasion rate vs MRW** as a function of Load Factor

It is important to note that these scanning rates map almost precisely to the undetected scanning rates observed in the naive experiments in Chapter V (see Figure 5.2d). The nature of the underlying algorithm of MRW — counting the number of destinations contacted — is inherently unavoidable by the worm. There is no way to cover up a connection to a destination. This contrasts directly with the results seen for the DSC detector. An evasive worm is able to attain a dramatically increased scanning rate than a naive worm because the evasive worm is able to avoid exhibiting the behavior that the detector relies on. That is not the case with the MRW worm, there is no way for the worm to avoid exhibiting the behavior of visiting destinations.

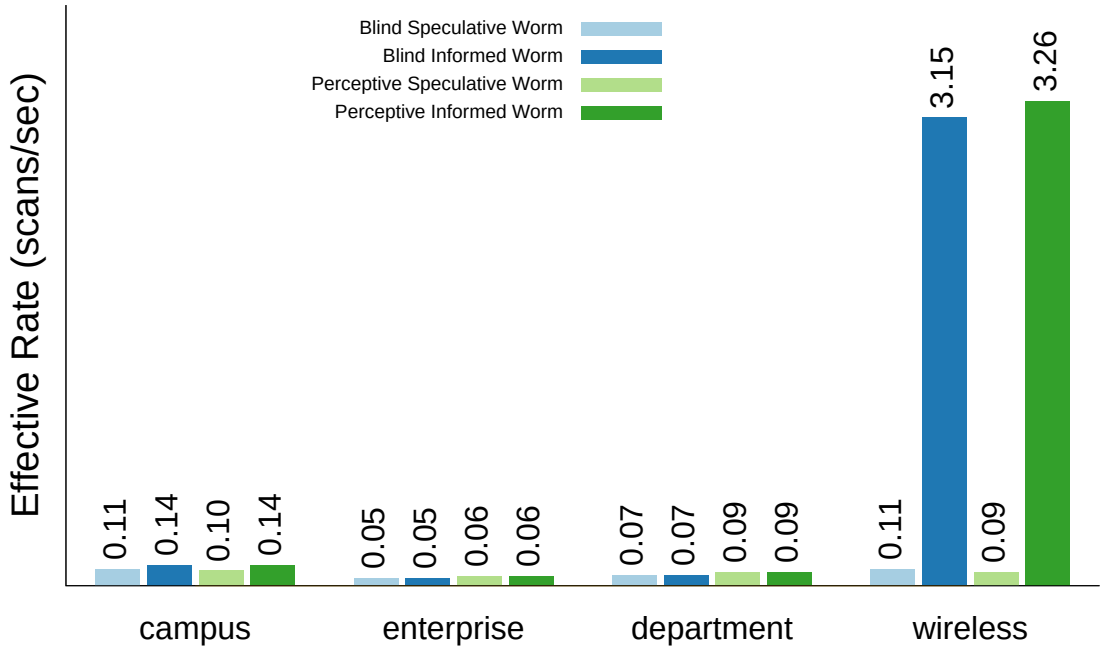


FIGURE 6.6. **Maximum effective rate vs MRW** as a function of Load Factor

RBS

The RBS detector also relies upon destination counting to detect worms. It uses a different detection algorithm than MRW, but our evasive worm implementation is essentially the same. Each instance of the worm runs an internal version of the RBS detector and monitors its own score. It determines the maximum sustained scanning rate that will not be flagged by the detector, and then scans at that rate. Listing 6.5 shows the implementation of the blind variants of the worm. Listing 6.6 shows the implementation of the perceptive variants.

Note that one significant distinction between the MRW and RBS detectors is that the RBS detector can identify a host as "legitimate" if it's connection activity does not look suspicious. This has the effect of resetting the host's score and

LISTING 6.5. **RBS Blind Evasive Worm**

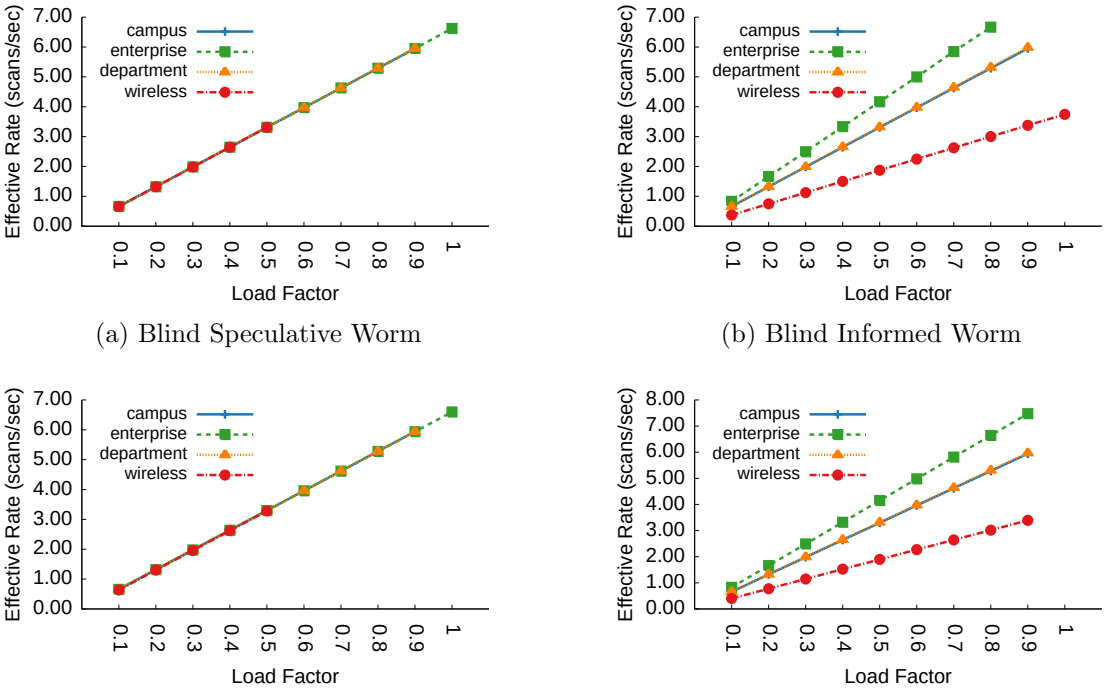
```
// interval is time between scans at maximum undetected scanning rate
function doScan() {
    wait(interval);
    scan();
}
```

LISTING 6.6. **RBS Perceptive Evasive Worm**

```
// interval is time between scans at maximum undetected scanning rate
function doScan() {
    wait(interval);
    updateScoreFromLegitimateActivity()
    if (! wouldRaiseAlarm()) {
        scan();
    }
}
```

starting it from zero again. It is possible that a worm that shaped its connection activity so as to periodically be identified as "legitimate" might be able to achieve a greater sustained scanning rate that one that constantly scans at an undetectable rate. Our simplistic evasive worm sufficiently showed that the RBS detector is outperformed by the MRW detector, so we did not pursue this avenue for further optimization.

Much like the other destination counting worms, Figure 6.7a shows that the blind speculative worms achieved scanning rate increases linearly with the load factor and is the same across all environments. The evasion rates achieved by the blind speculative worm is quite different however (Figure 6.8a). The worm has difficulty consistently evading the detector in the campus environment. The reason for that is that the campus environment has the most restrictive thresholds for the RBS detector, so the speculative worm is actually operating quite close the actual



(a) Blind Speculative Worm

(b) Blind Informed Worm

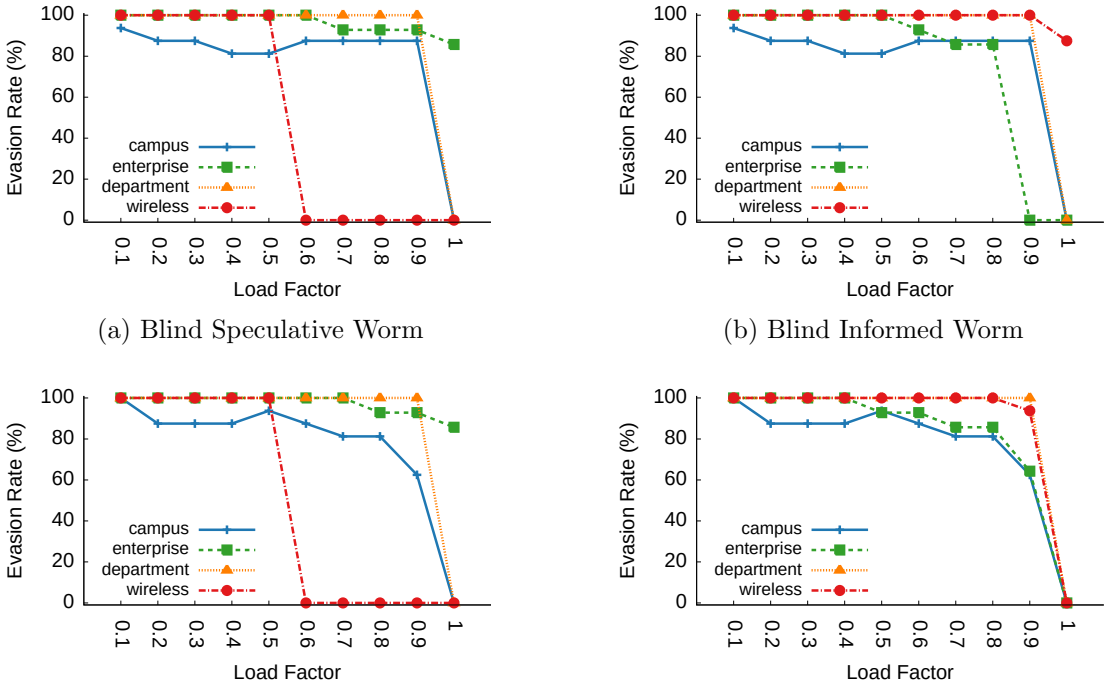
(c) Perceptive Speculative Worm

(d) Perceptive Informed Worm

FIGURE 6.7. **Effective scanning rate vs RBS** as a function of Load Factor

threshold for the environment. This graph may look odd compared with the success of the naive worms at evading the detector in all environments (see Figure 5.2c). It is important to keep in mind that this graph is a function of load factor, which is quite different than the graph from Chapter V which is a function of scanning rate. In this case, the worm is substantially more aggressive than the naive worm, with the achieved rate of greater than one scan per second even at a load factor of 0.1.

The RBS detector shows a poor maximum effective rate across the board. It allows an evasive worm to operate freely at relatively high scanning rates (Figure 6.9).



(a) Blind Speculative Worm

(b) Blind Informed Worm

(c) Perceptive Speculative Worm

(d) Perceptive Informed Worm

FIGURE 6.8. **Evasion rate vs RBS** as a function of Load Factor

PGD

The PGD detector is the final detector that uses some form of destination counting as its sole detection heuristic. It is also the only detector that measures the aggregate traffic from a protected domain. Like the other destination counting detectors, there is only one way to evade detection, and that is to scan at the maximum sustained rate that won't be detected.

The implementation of the PGD evasive worm is very similar to the implementations of the other destination counting evasive worms. The blind variants don't bother to run an implementation of the detector internally because it isn't useful. Without observing other traffic, an internal implementation of PGD

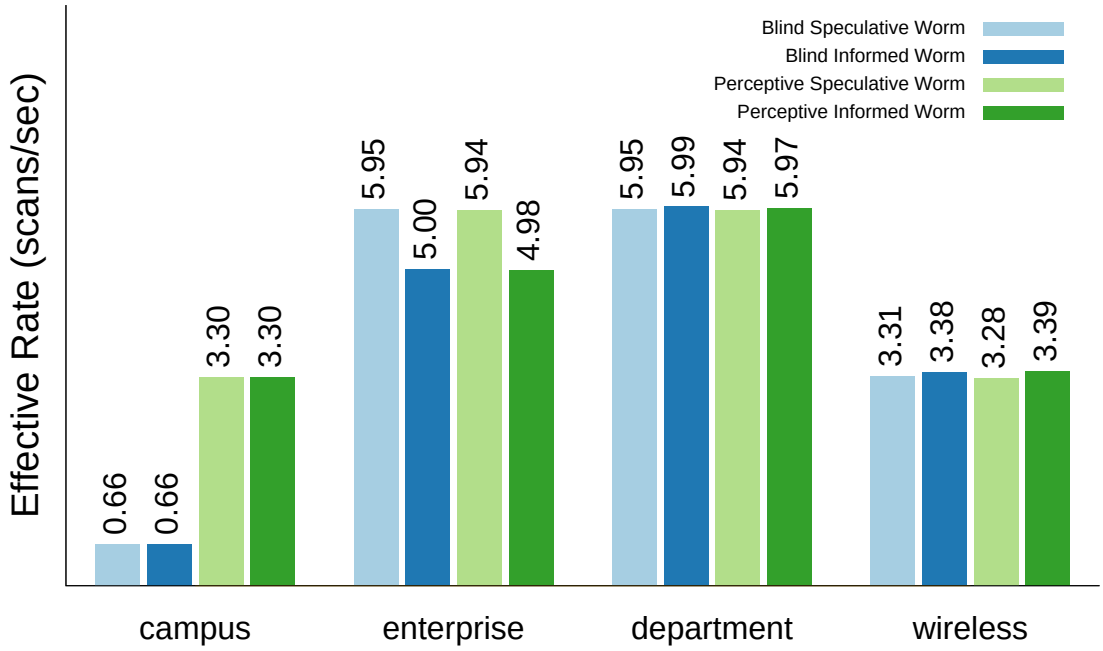


FIGURE 6.9. **Maximum effective rate vs RBS** as a function of Load Factor

would do nothing other than to limit the scanning rate to the maximum undetected rate. This can already be accomplished by simply calculating that rate from the speculated thresholds. Listing 6.7 shows the implementation of the blind variants of the worm. The perceptive version of the implement an internal version of PGD and can stop scanning during a detection window when the detector is getting close to the its threshold. Listing 6.8 shows the implementation of the perceptive variants.

Figure 6.10a and Figure 6.11a show the very poor performance of the blind speculative version of PGD evasive worm (note that an effective rate is only reported for scenarios where at least one infected host evaded detection). The blind speculative worm never evaded detection at load factors of greater than 0.5, and was limited to effective rates of 0.3 scans per second or less in all scenarios. This

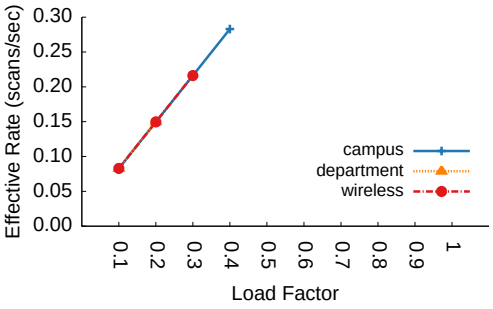
LISTING 6.7. **PGD Blind Evasive Worm**

```
// interval is time between scans at maximum undetected scanning rate
function doScan() {
    if (now + interval > nextWindowStartTime()) {
        scheduleScheduleScanFor(nextWindowStartTime());
    } else {
        scheduleScanFor(now + interval);
    }
}
```

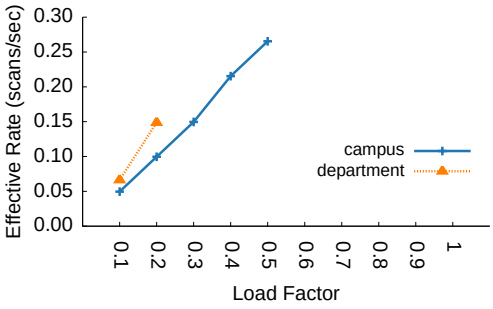
LISTING 6.8. **PGD Perceptive Evasive Worm**

```
function doScan() {
    scoreLegitimateConnections();
    while (wouldScanRaiseAlarm()) {
        waitOneTick();
    }
    // how much time is left in this detection window?
    timeRemainingInWindow = windowEndTime - lastActiveTime;
    // how fast should we scan to use remaining available
    // slots in window? this keeps us from scanning too fast
    targetInterval = timeRemainingInWindow / countScansUnderThreshold();
    scheduleScanFor(lastActiveTime + targetInterval);
}
```

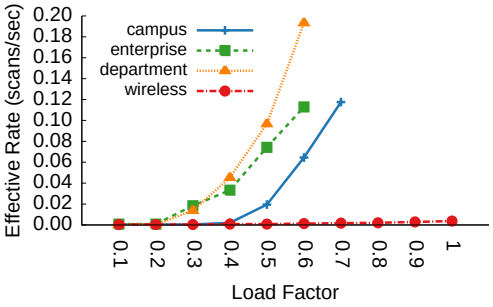
is because the infected host cannot see any other traffic in the domain so assumes that it is the only host generating traffic. It generates enough traffic to fully fill the threshold of the detector (limited by the load-factor). For detectors that only operate on the traffic from a single host, there is a reasonable chance that each individual host won't generate much legitimate traffic, so that a blind worm can operate at higher load factors without triggering the detector. PGD operates on the aggregate of traffic from the protected domain, so there is less overall variation in the legitimate traffic, and the proportion of the traffic from a single host to the overall threshold is substantially lower. This leads the blind worm to trip the alarm at very low load factors.



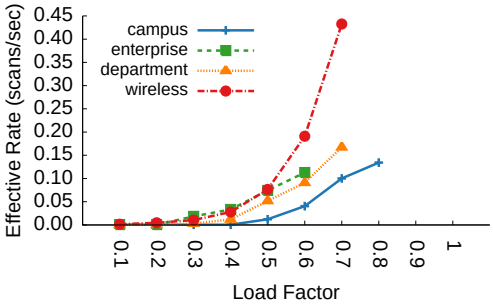
(a) Blind Speculative Worm



(b) Blind Informed Worm



(c) Perceptive Speculative Worm

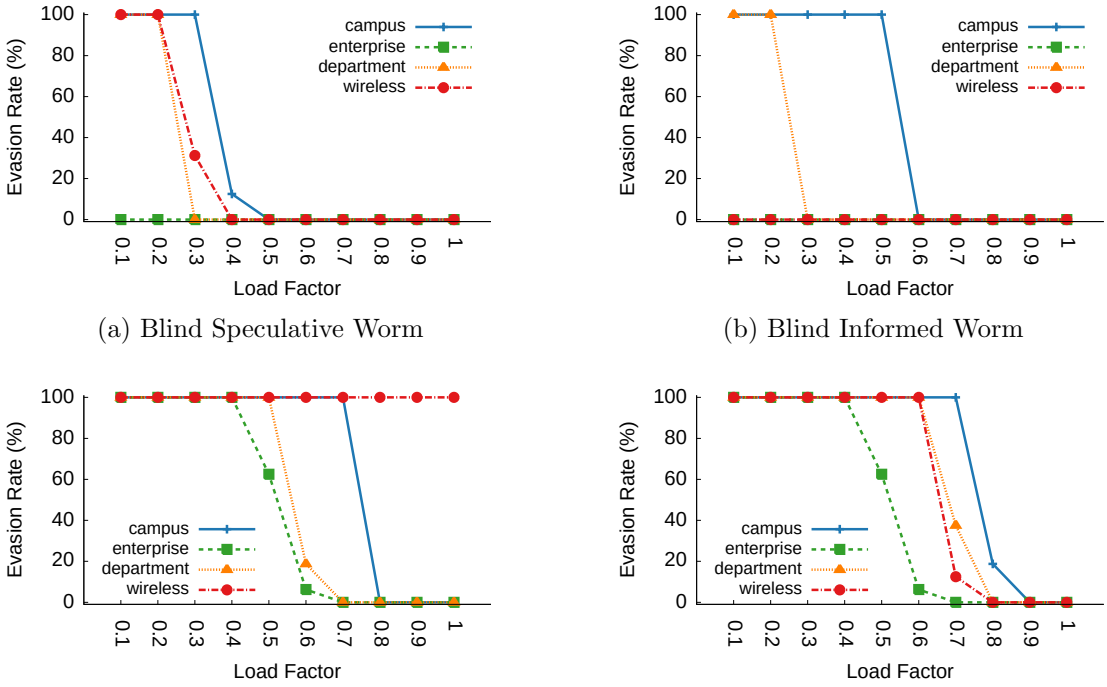


(d) Perceptive Informed Worm

FIGURE 6.10. **Effective scanning rate vs PGD** as a function of Load Factor

A second interesting feature of these results is that the worm was unable to evade detection in any scenario in the enterprise environment. This seems odd given that the that detector showed greater than 80% false negatives at the slowest scanning rate against a naive worm (see Figure 5.2f). The reason for this is that the PGD detector operates on distinct one-minute intervals of network traffic and in the enterprise environment there is an interval that is exactly one connection below the threshold. A worm scanning at 0.005 scans per second makes a scan only once every 3.33 minutes. The naive worms had a decent shot at simply skipping over that specific interval. However, the evasive worms attempt to maximize their scanning rate so perform at least one scan every minute. This always triggers the

detector in the campus environment. The other environments do not have any windows that are so close to the threshold with just the legitimate traffic, so the worm is able to make at least some scans. TODO: add some analysis on the chance of windows like this. It seems like they should be rare.



(a) Blind Speculative Worm

(b) Blind Informed Worm

(c) Perceptive Speculative Worm

(d) Perceptive Informed Worm

FIGURE 6.11. **Evasion rate vs PGD** as a function of Load Factor

The blind informed version of this worm performs even worse (see Figure 6.10b and Figure 6.11b). The worm is still hampered by not seeing any of the traffic, but now is using more aggressive thresholds. This makes the worm more likely to fail, and in fact in the wireless environment it now no longer is ever able to evade detection.

The perceptive variants of the worm perform much better. The perceptive

speculative version of the worm is able to evade detection in all environments up to a load factor of 0.4 (see Figure 6.11c). This enables it to achieve decent scanning rates in all environments (Figure 6.10c). The perceptive informed worm does even better (see Figure 6.11d and Figure 6.11c). It has lower evasion rates at high load factors because it is using higher threshold targets, but gets correspondingly higher effective rates.

The maximum evasive rates achieved against the PGD detector (Figure 6.12) show that PGD a very effective detector at suppressing worm traffic. In several scenarios in both the enterprise and wireless environments the evasive worms were unable to evade detection in any scenarios. An interesting development is that in the campus and department environments the perceptive worms performed more poorly than the blind worms. It is important to remember that the PGD detector operates on distinct one minute segments of network traffic. In these cases, the perceptive worms are cautious in an attempt to evade detection. They limit their scanning rate to a fraction of the available scans under the threshold because they do not know the actual timing of the windows of the detector. The blind worms don't have this caution and essentially luck out by scanning more aggressively. It seems quite likely that further optimizations to the perceptive worm would improve its performance to match that of the blind variant in those environments.

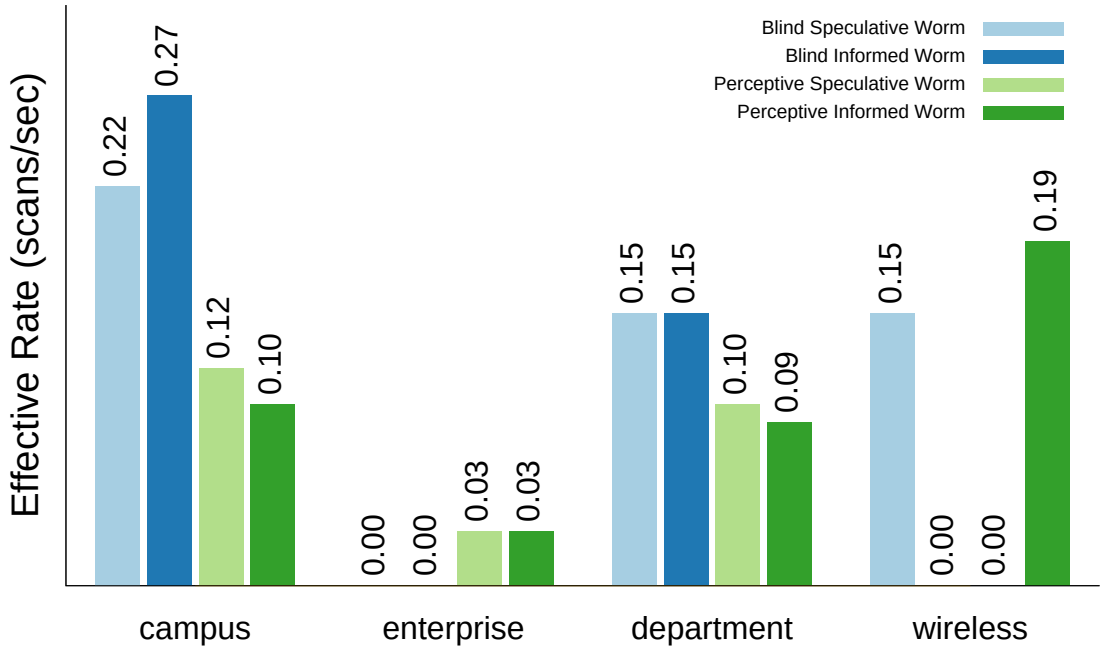


FIGURE 6.12. **Maximum effective rate vs PGD** as a function of Load Factor

TRW

As we observed in Chapter V, a worm that can make connections to hosts known to be running the target service of the worm can evade the TRW detector. TRW detects scanning behavior by watching for first contact connections that fail, but its scoring algorithm uses both successful connections and unsuccessful connections in assessing a host, where a successful connection essentially cancels out an unsuccessful connection. If a malicious user wanted to release a worm to evade the TRW detector, a sensible strategy would be to set up a series of hosts across the Internet running the target service. Each instance of the TRW worm could make periodic connections to these hosts to keep its score from exceeding the TRW threshold. Furthermore, TRW only tracks a connection as a "first contact"

for 30 minutes, so the dummy hosts could be reused periodically. In evaluating the TRW detector we consider two scenarios. In the first scenario each worm has 100 of these *known neighbors* at its disposal; in the second scenario each worm has 1000 known neighbors.

Listing 6.9 shows the core implementation of the TRW Blind Evasive worm. It runs an instance of the TRW detector internally and keeps a running tally of its score. It it appears that a connection will cause the TRW detector to raise the alarm, a known neighbor is used instead to reduce the score. If the worm doesn't have a known neighbor that can be used (all the known neighbors have been used more recently than the TRW timeout) the worm will wait until one of the known worms can be used again and will appear as a new destination to TRW. The Perceptive variant of the worm (Listing 6.10 is quite similar, the only being that the perceptive worm can more accurately track its score.

For the Blind Speculative variant of the worm with 100 known neighbors the worm is able to achieve scanning rates between 0.10 and .13 scans per second (Figure 6.13a). The increase is linear with the load factor. This variant of the worm evades detection almost all scenarios (Figure 6.14a).

The perceptive speculative variant of the worm shows very similar performance. It is able to achieve a higher effective scan rate in the wireless environment because the higher volume of successful legitimate connections allows it to complete more random scans without tripping the detector (Figure 6.14c).

LISTING 6.9. **TRW Blind Evasive Worm**

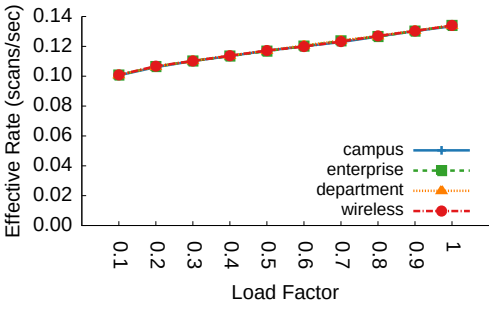
```
function doScan() {
    if (wouldFailedConnectionRaiseAlarm()) {
        targetAddress = waitForAvailableKnownNeighbor(now);
        scanTargetAddress(targetAddress);
    } else {
        scanRandomAddress();
    }
    tick++;
}

// get a known neighbor, wait past the TRW timeout
// so it appears new again if needed
function waitForAvialableKnownNeighbor(now) {
    neighbor = knownNeighbors.getFirst();
    if (neighbor.lastVisitedTime + trwTimeout > now) {
        now = waitUntil(neighbor.lastVisitedTime + trwTimeout);
        neighbor.lastVisitedTime = now;
        knownNeighbors.addLast(neighbor);
    }
    return neighbor.address;
}
```

The informed variants of the worm show slightly higher effective rates (Figure 6.14b and Figure 6.14d). As the evasive worms increase their scanning rate they are less able to evade detection, but even in the worst case are still evading detection more than 50% of the time (Figure 6.14).

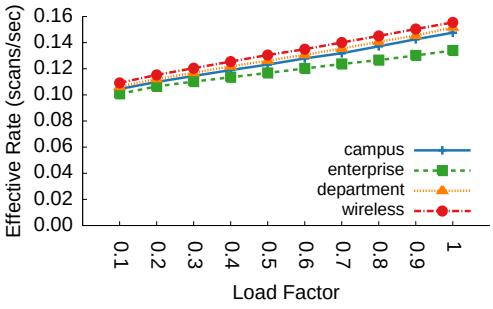
We reran the same experiments with 1000 known neighbors (Figure 6.15 and Figure 6.16). The results are largely similar except that the effective rates attained by the evasive worms increased by roughly a factor of 10.

The evasive worms with 1000 neighbors had similar evasion rates to the worms with only 100 neighbors, but were able to accomplish significantly more scanning.

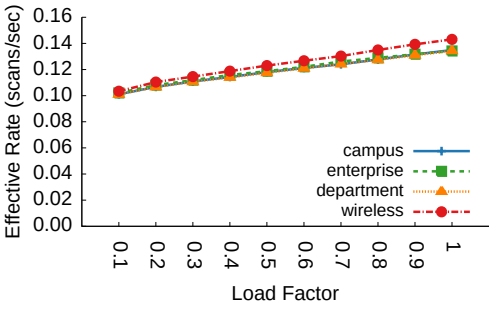
Figure 6.17 shows the maximum effective rate of each variant of this worm in



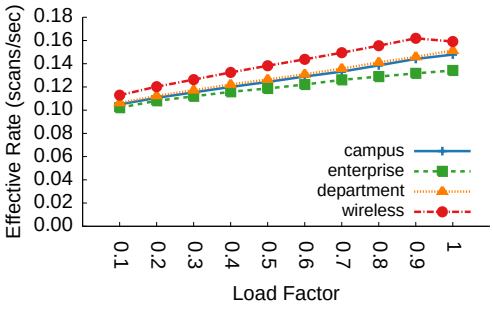
(a) Blind Speculative Worm



(b) Blind Informed Worm

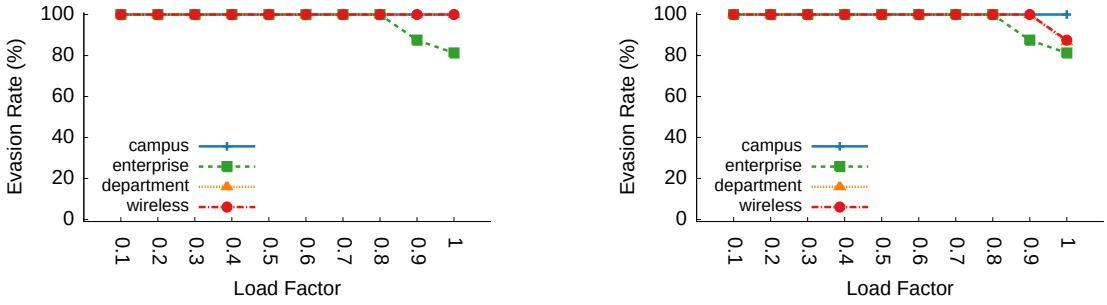


(c) Perceptive Speculative Worm



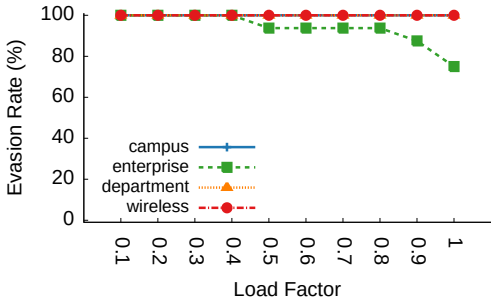
(d) Perceptive Informed Worm

FIGURE 6.13. **Effective scanning rate vs TRW with 100 known targets** as a function of Load Factor

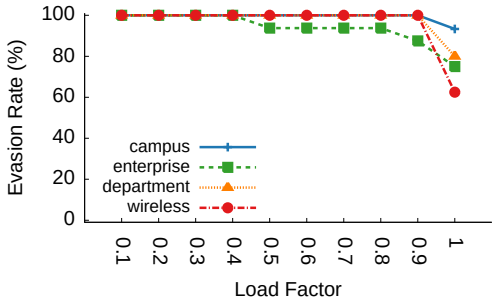


(a) Blind Speculative Worm

(b) Blind Informed Worm

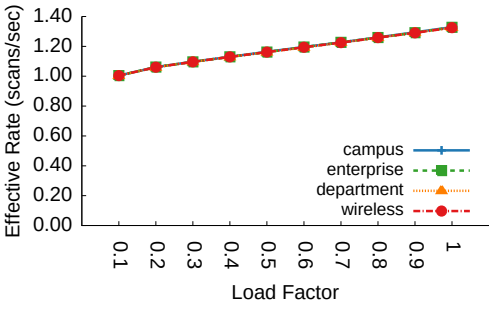


(c) Perceptive Speculative Worm

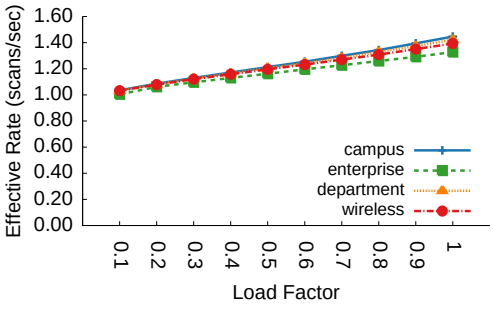


(d) Perceptive Informed Worm

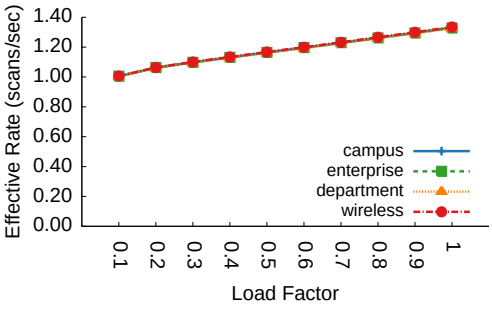
FIGURE 6.14. **Evasion rate vs TRW with 100 known targets** as a function of Load Factor



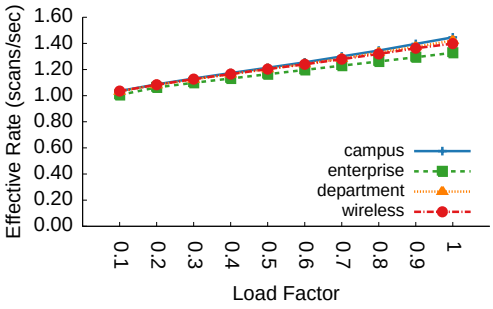
(a) Blind Speculative Worm



(b) Blind Informed Worm

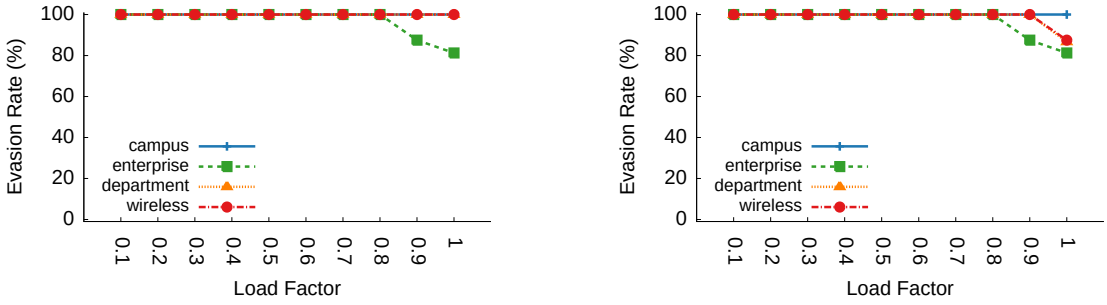


(c) Perceptive Speculative Worm



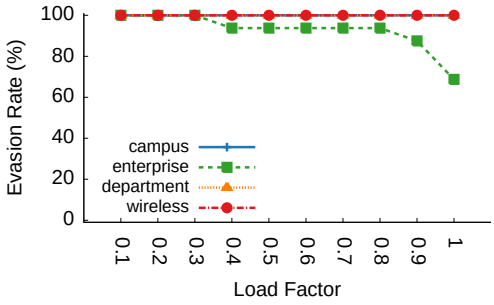
(d) Perceptive Informed Worm

FIGURE 6.15. **Effective scanning rate vs TRW with 1000 known targets** as a function of Load Factor

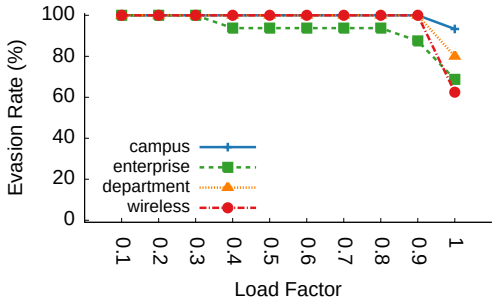


(a) Blind Speculative Worm

(b) Blind Informed Worm



(c) Perceptive Speculative Worm



(d) Perceptive Informed Worm

FIGURE 6.16. **Evasion rate vs TRW with 1000 known targets** as a function of Load Factor

LISTING 6.10. **TRW Perceptive Evasive Worm**

```
function doScan() {
    scoreLegitimateConnections();
    if (wouldFailedConnectionRaiseAlarm()) {
        targetAddress = waitForAvailableKnownNeighbor(now);
        scanTargetAddress(targetAddress);
    } else {
        scanRandomAddress();
    }
    tick++;
}

// get a known neighbor, wait past the TRW timeout
// so it appears new again if needed
function waitForAvialableKnownNeighbor(now) {
    neighbor = knownNeighbors.getFirst();
    if (neighbor.lastVisitedTime + trwTimeout > now) {
        now = waitUntil(neighbor.lastVisitedTime + trwTimeout);
        neighbor.lastVisitedTime = now;
        knownNeighbors.addLast(neighbor);
    }
    return neighbor.address;
}
```

each environment. The most striking feature of this graph is that the evasive worm is able to achieve scanning rates more than 20 times greater that the fastest naive worm (please refer back to Figure 5.2a). This shows that the underlying mechanism used by TRW is susceptible to evasion.

TRWRBS

Because it relies on the TRW scoring mechanism for part of its detection algorithm, it is again vulnerable to spoofing with known neighbors. We therefore evaluate two versions of this worm, one with 100 known neighbors that can be used to spoof TRW, and one with 1000 known neighbors. The TRWRBS evasive worm is among the most complex of the evasive worms shown evaluated. It cannot

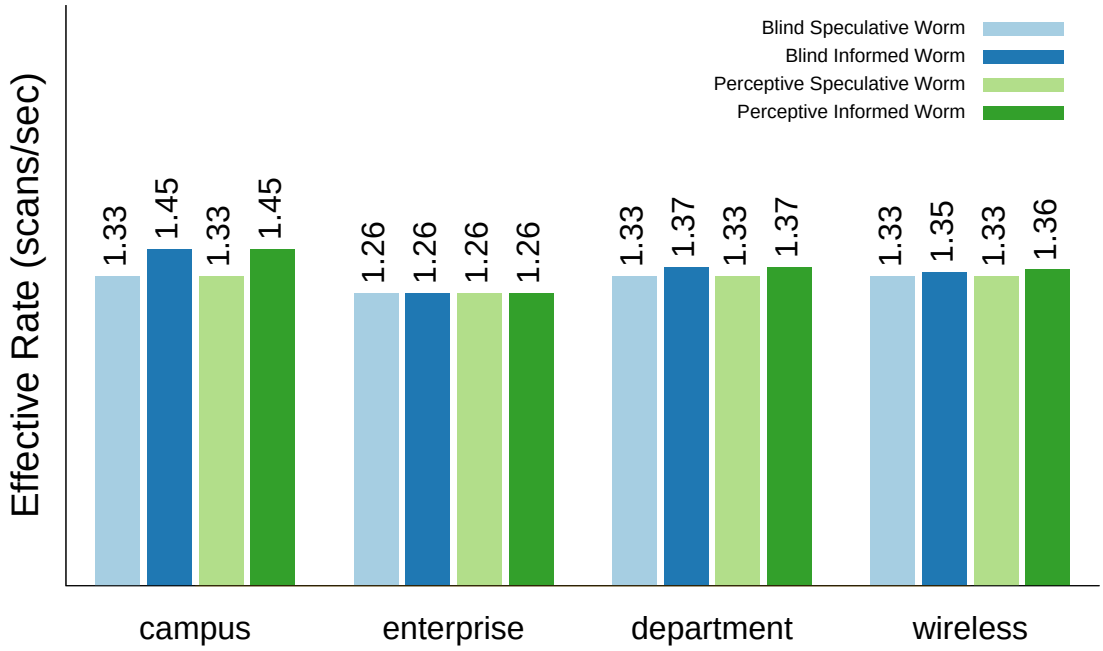


FIGURE 6.17. **Maximum effective rate vs TRW** as a function of Load Factor

simply schedule connections at a set rate because it must manage its list of known neighbors to avoid connection failure detection. It also can use the score of either detection heuristic to balance out the score of the other, so must also manage the balance between the two scores.

Listing 6.11 shows the implementation of the blind variants of the worm. When it is time to scan, the worm determines whether scanning would raise the alarm or whether its TRW score is getting excessive. If it has a known neighbor available that will help its score, it uses it. Otherwise it waits a tick and re-examines its scores. Waiting will help to reduce the RBS score (as it increase the time interval between scans) and will also potentially allow a known neighbor to become available.

LISTING 6.11. **TRWRBS Blind Evasive Worm**

```
function doScan() {
    while (wouldFailedConnectionRaiseAlarm() ||
           predictedTRWScore >= etaOne) {
        if (predictedTRWScore >= etaOne &&
            knownNeighborIsAvailable(now)) {
            // okay, we've got a bad TRW score and there is a neighbor
            // avialable now if using that neighbor would improve our
            // score, then let's do it
            predictTRWScoreAfterSuccessfulFirstContact();
            if (predictedTRWScore * predictedRBSScore < etaOne) {
                scanTargetAddress(getAvailableKnownNeighbor(now));
                return;
            }
            // if it doesn't improve things enough, then let's wait
            // and see what happens
        } else if (knownNeighborIsAvailable(now)) {
            // our overall score is too high, but our predicted
            // TRW score isn't so bad
            predictTRWScoreAfterSuccessfulFirstContact();
            if (predictedTRWScore * predictedRBSScore < etaOne) {
                scanTargetAddress(getAvailableKnownNeighbor(now));
                return;
            }
        }
        // else let's wait a tick to improve our RBS score or
        // wait for a known neighbor to become available
        tick++;
    }

    // a random scan wouldn't cause a problem
    return randomScan();
}
```

Listing 6.11 shows the implementation of the perceptive variants of the worm. These versions of the worm are very similar, but add additional two checks against legitimate traffic. The first check comes before any scan attempt is considered (to account for any legitimate traffic since any previous scan) and the second comes after every wait. Waiting may allow for legitimate traffic to change the hosts score which will change the decision on whether to scan or not.

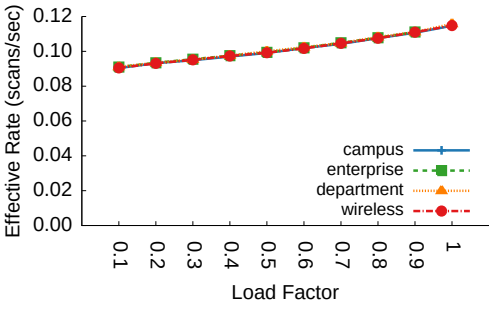
Figure 6.18a shows the effective rates achieved by the 100-neighbor blind speculative variant of the worm. The worm does not scan quickly in comparison to the RBS evasive worm, and is extremely effective at evading detection (Figure 6.19a. This may indicate that the evasive worm implementation is not aggressive enough at scanning and could be improved.

The results for the perceptive speculative variant of the worm backs up this hypothesis. They show slower scan rates than the blind variant of the worm. Accounting for the legitimate traffic from the host is causing the worm to scan more cautiously. The complexity of balancing the two detection heuristics allows for considerable room to vary the implementation of the worm, unlike the much simpler worms used to evade destination counting heuristics like MRW, RBS, and PGD.

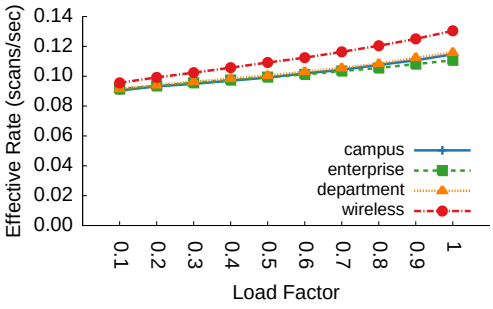
The blind informed variant of the worm attains greater effective rates in most environments as it uses more accurate thresholds (Figure 6.18b). It maintains high evasion rates with the higher scan rates as seen in Figure 6.19b.

The perceptive informed worm performs similarly (Figure 6.18d and Figure 6.19d) but again shows slightly slower scan rates than the blind worm.

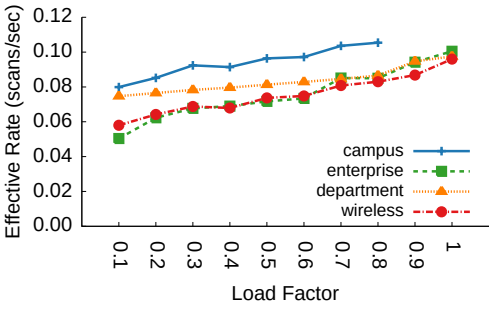
Increasing the known neighbor count of the worm to 1000 increases its effective scan rate by approximately a factor of 10 at high load factors. Figure 6.20a shows the effective rates attained by the blind speculative variant of the worm with 1000 neighbors. It is interesting to note that at low load factors,



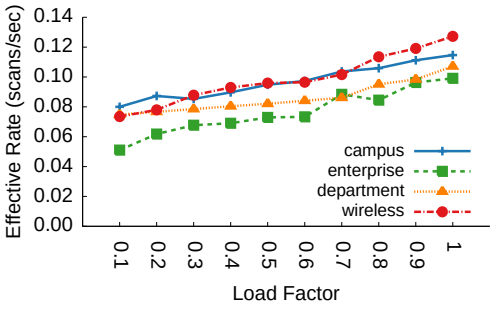
(a) Blind Speculative Worm



(b) Blind Informed Worm

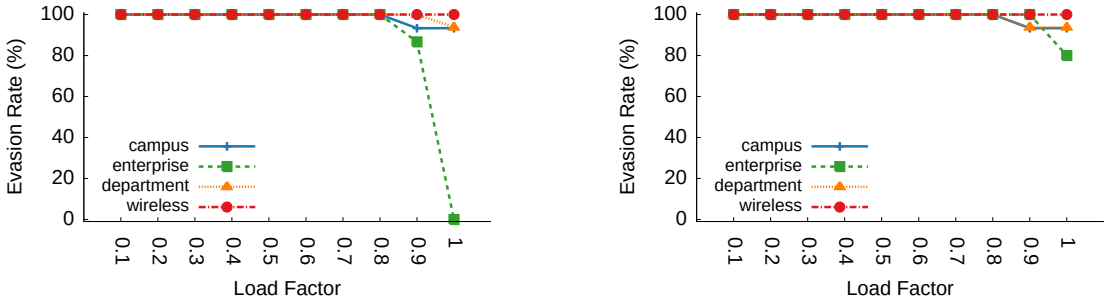


(c) Perceptive Speculative Worm



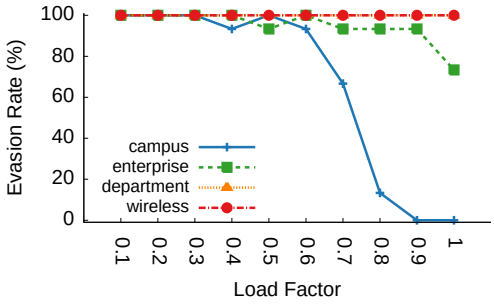
(d) Perceptive Informed Worm

FIGURE 6.18. **Effective scanning rate vs TRWRBS with 100 known targets** as a function of Load Factor

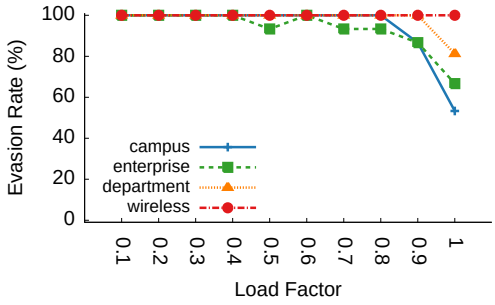


(a) Blind Speculative Worm

(b) Blind Informed Worm

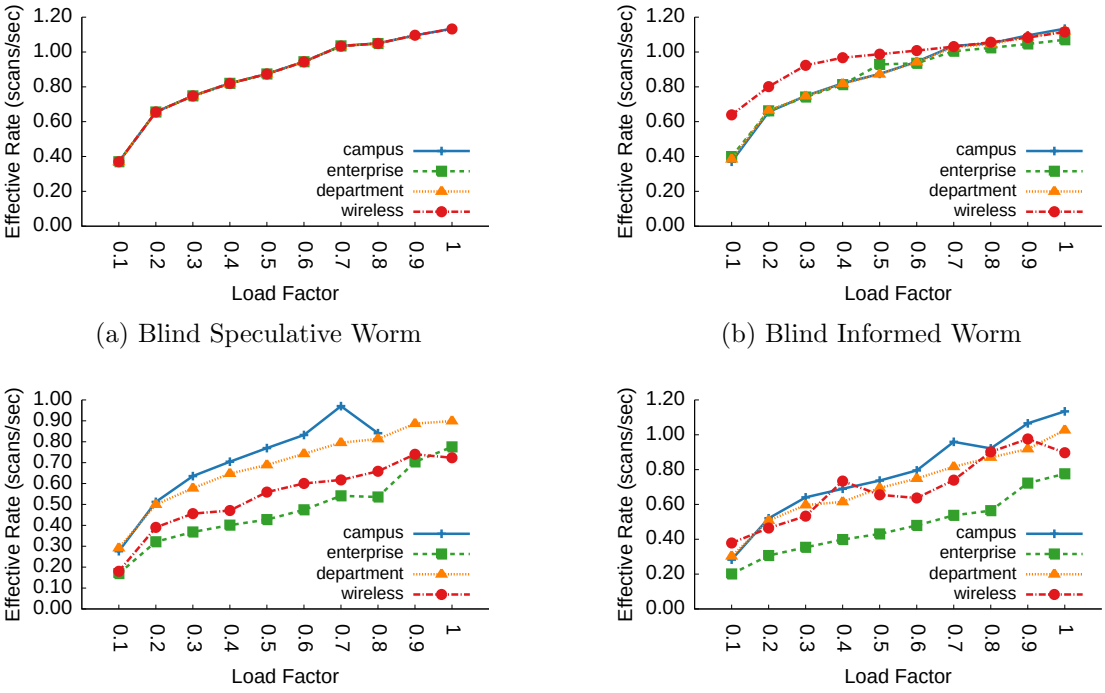


(c) Perceptive Speculative Worm



(d) Perceptive Informed Worm

FIGURE 6.19. **Evasion rate vs TRWRBS with 100 known targets** as a function of Load Factor



(a) Blind Speculative Worm

(b) Blind Informed Worm

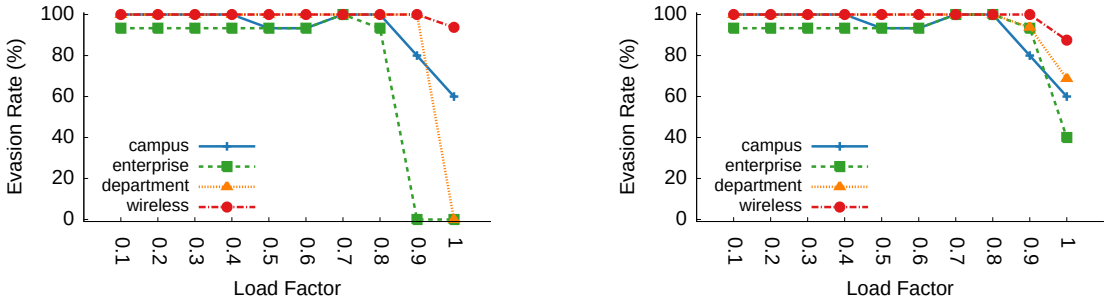
(c) Perceptive Speculative Worm

(d) Perceptive Informed Worm

FIGURE 6.20. **Effective scanning rate vs TRWRBS with 1000 known targets** as a function of Load Factor

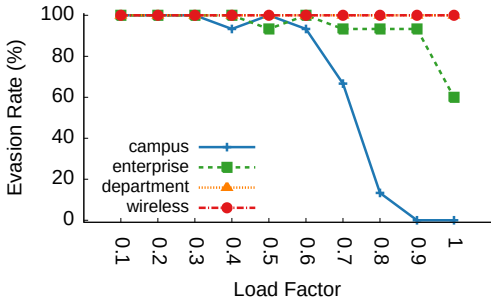
the worm does achieve a 10x increase over its performance with only 100 neighbors. This is because the RBS component of the detector limits the overall scanning rate at low load factors so it is not able to use all of its known neighbors. At a load factor of 0.1, the blind speculative worm uses less than 900 of its known neighbors. The experiments are long enough in duration that neighbors that are used early during the experiment are able to be re-used, so it is actually effectively using less than half of its neighbors. At a load factor of 1.0 the worms are able to achieve over 2000 scans, using all of their known neighbors. At the higher load factors the TRW portion of the algorithm seems to be more limiting than the RBS portion.

Examining the maximum effective rate achieved by each variant of the

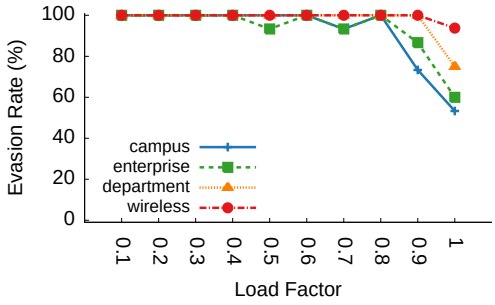


(a) Blind Speculative Worm

(b) Blind Informed Worm



(c) Perceptive Speculative Worm



(d) Perceptive Informed Worm

FIGURE 6.21. **Evasion rate vs TRWRBS with 1000 known targets** as a function of Load Factor

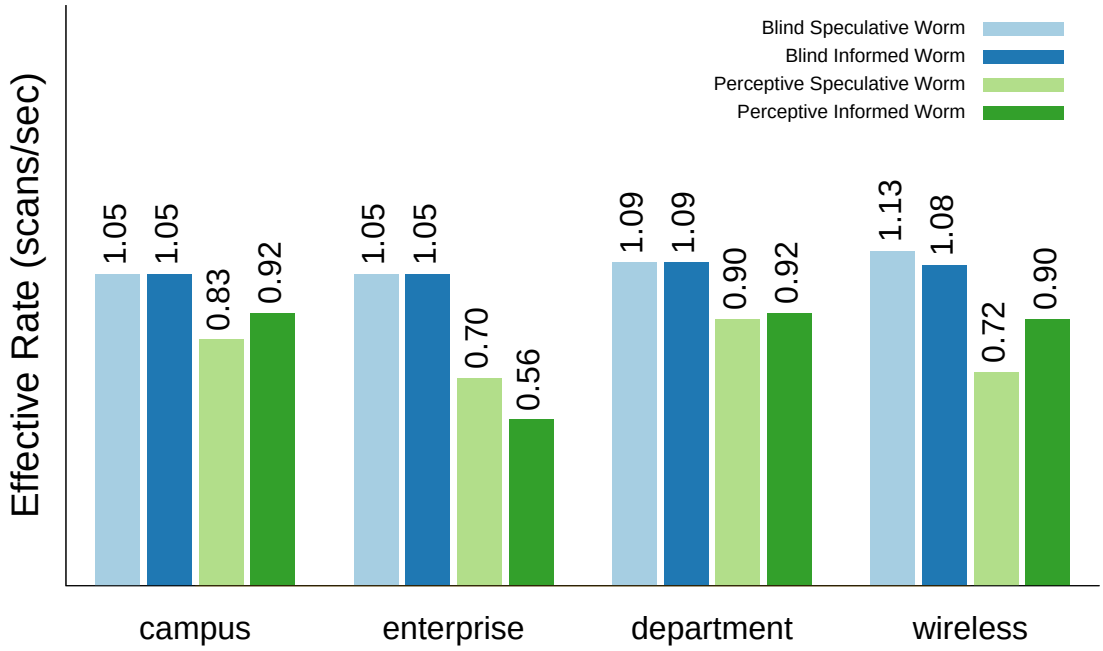


FIGURE 6.22. **Maximum effective rate vs TRWRBS** as a function of Load Factor

TRWRBS evasive worm in every environment in Figure 6.22, we can see that that the TRWRBS detector is relatively ineffective at suppressing worm scanning activity. In all scenarios the worm was able to achieve an effective rate of better than 0.5 scans per second without being detected. The fact that the perceptive variants of the worm performed more poorly than the blind version likely indicates a failure to effectively code the evasive worm. The complexity of the code required to evade a detector such as TRWRBS does play in its favor however, as it raises the bar for what a malicious entity needs to do to evade the detector.

LISTING 6.12. **TRWRBS Perceptive Evasive Worm**

```
function doScan() {
    scoreLegitimateConnections();
    while (wouldFailedConnectionRaiseAlarm() ||
           predictedTRWScore >= etaOne) {
        if (predictedTRWScore >= etaOne &&
            knownNeighborIsAvailable(now)) {
            // okay, we've got a bad TRW score and there is a neighbor
            // available now if using that neighbor would improve our
            // score, then let's do it
            predictTRWScoreAfterSuccessfulFirstContact();
            if (predictedTRWScore * predictedRBSScore < etaOne) {
                scanTargetAddress(getAvailableKnownNeighbor(now));
                return;
            }
            // if it doesn't improve things enough, then let's wait
            // and see what happens
        } else if (knownNeighborIsAvailable(now)) {
            // our overall score is too high, but our predicted TRW
            // score isn't so bad
            predictTRWScoreAfterSuccessfulFirstContact();
            if (predictedTRWScore * predictedRBSScore < etaOne) {
                scanTargetAddress(getAvailableKnownNeighbor(now));
                return;
            }
        }
        // else let's wait a tick to improve our RBS score or
        // wait for a known neighbor to become available or
        // for legitimate traffic to improve our score
        tick++;
        // score any new legitimate connections that happen
        // while we are waiting
        scoreLegitimateConnections();
    }

    // a random scan wouldn't cause a problem
    return randomScan();
}
```

CHAPTER VII

A NEW WORM DETECTOR

The definition of a network worm is code that scans the network to find and infect new hosts. With this definition, the only truly fundamental behavior of worms is that of connecting to new destinations. Behavior based detection systems that do not focus on this one fundamental behavior can be evaded successfully by sufficiently smart worms. This conclusion was validated by the results in Chapter VI. Therefore, the one behavior that we must include to improve worm detection is that of visiting destinations. Are there techniques for detecting anomalous destination visiting patterns that have not been explored yet and that could help to reduce the effective scanning rate of worms?

We approach the issue from several directions. First, we investigate a new way of preventing fast scanning by using a burst detection algorithm with more detailed thresholds than existing algorithms. Second, we ensure that quiescent periods in network activity do not disappear. Legitimate network traffic from most hosts is not as consistently persistent as worm scanning, and by monitoring the connection behavior of a host we can prevent a worm from constantly scanning for targets. Finally, rather than applying a single threshold to an entire network of hosts, we group hosts based on their recent activity profile and establish different thresholds for different groups of hosts.

These techniques enable us to build a worm detector with superior performance characteristics to the the detectors evaluated in Chapters V and VI. We present the design and implementation of the detector in the remainder of this chapter, and evaluate its performance against both naive and evasive worms in Chapter VIII.

7.1. Preventing Fast Scanning

The most fundamental behavior of self-propagating network scanning worms is the behavior of contacting new destinations seeking new victims to infect. This behavior simply cannot be avoided by a worm that is looking to propagate, so looking for anomalies in the rate at which new destinations are contacted (*first-contact connections*) is an important component of a detection system. The MRW and RBS detectors are two important systems that rely on this heuristic. However, we have found that it is possible to improve their performance, achieving fewer false negatives for a given false positive rate.

The MRW detector counts the number of first-contact connections in a series of time windows of different sizes (hence the "multi-resolution" in the methods's title). It uses a relatively small set of windows (typically fewer than 10). An intermediate window size might produce a detection window that would detect a worm more quickly than the bigger or smaller sizes in use, but due to the limited number of widows, the detector cannot take advantage it.

RBS, on the other hand, computes a threshold for every different window

size (using the number of connections instead of time to describe the window, with elapsed time from the first connection to the last acting as the threshold). This yields more opportunities to detect worms of different rates more quickly, as we effectively use all the different window sizes. However, RBS attempts to fit a single curve to the distribution of inter-connection intervals and uses this curve to generate the thresholds. This leads to sub-optimal thresholds as in practice the distribution of inter-connection intervals does not map perfectly to a simple curve.

Our approach avoids the drawbacks in the MRW and RBS approaches. We have developed a new approach to monitoring for anomalies in the number of destinations contacted. Rather than using fixed time-limited window sizes, we use RBS's method of creating a window for every different size of connection burst. There is a threshold for a 2-connection burst, a 3-connection burst, a 4-connection burst and so on up to a maximum burst size. During the training phase, we measure the durations observed for each burst size and base our threshold on the minimum duration observed for a given burst size. This has the advantages of giving us a threshold for every different size burst (a large number of window sizes) while allowing for a more complex distribution of inter-connection interval times (more accurate thresholds). In our evaluations this new method yields better performance against slow scanning worms.

The drawback to this method is greater overhead for storing different thresholds and greater computational requirements for examining a recent

connection history to determine if it violates any of the thresholds. However, a truism is that computational power and storage space are constantly increasing, and we feel that this additional load is relatively inconsequential. Further research remains to be on the performance aspect of this approach.

7.2. Quiescent Periods

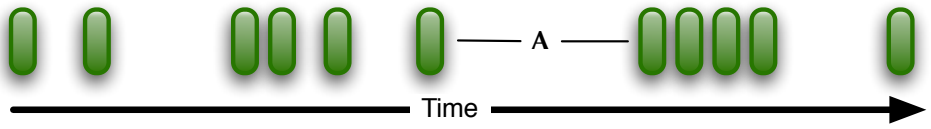
To evade a detector using an approach like the one above, a worm must limit the first-contact rate of the host to some value lower than the detection threshold. As a host makes bursts of legitimate first-contact connections, the worm should go quiet to avoid exceeding the detection threshold. When the host is otherwise idle, however, the worm is free to make first contact connections without danger of exceeding the threshold.

Legitimate traffic is typically bursty, with first-contact connections occurring in groups and quiet periods between them (see Figure 7.1a for an example). A worm that scans at a fixed rate will make connections during the middle of a legitimate burst which will raise the overall observed connection rate from the host (Figure 7.1b). An adaptive worm can avoid this additive effect by only scanning when the host is otherwise idle. This will reduce the rate of connections as it will keep the worm connections from adding to the legitimate connections (Figure 7.1c). This is the technique used by the rate adaptive evasive worms evaluated in Chapter VI. Preventing or limiting this behavior would help to reduce the achievable scan rate of a worm, and is the basis for our next detection heuristic.

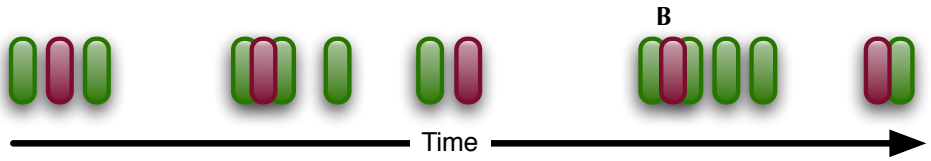
A normal host will exhibit regular *quiescent* periods where it does not make any first-contact connections. Absence of these quiescent periods may be an indicator that the host is infected by a worm that is scanning the network..

Figure 7.1 shows a series of example connection patterns that illustrate this detection mechanism. Figure 7.1a shows an example pattern of legitimate connections. Point **A** in the figure shows a quiescent period with no worm traffic, followed by a burst of connections. Figure 7.1b shows the legitimate traffic with the addition of naive worm traffic. Point **B** indicates a spot of increased connection rate due to the worm connections adding to the burst of legitimate traffic. An adaptive worm can avoid such a burst though. Figure 7.1c shows the legitimate traffic with an adaptive worm overlaid. Even at a higher scanning rate than the naive worm (with eight worm scans instead of five) the adaptive worm avoids any scanning rate greater than the legitimate traffic for small window sizes. However, it also does not exhibit any quiescent periods at all, which indicates the presence of the worm.

The QPD detection heuristic measures the duration of active periods where a host is making first-contact connections separated by quiescent periods of a fixed size where no first-contact connections occur. For a given quiescent period size, it measures the mean and standard deviation of all the active periods separated by such a period. These values are used to generate a threshold duration. If a host has an active period exceeding this duration, it is likely infected with a worm.



(a) Legitimate Connections



(b) Legitimate Connections Plus Naive Worm Connections

(c) Legitimate Connections Plus Rate Adaptive Worm Connections

FIGURE 7.1. **Examples of observed connections over time**

7.3. Clustering

Existing behavior-based detection systems employ the same threshold for all hosts in a protected network. This is a poor choice because the hosts in a network show widely divergent behaviors. Desktop computers used primarily for web surfing make connections in a different pattern than a department email server would, for example. If a desktop computer started making connections at the same rate as the email server it is likely an anomalous event, something strange must have happened to that computer. But if the desktop computer has same thresholds applied to it that the email server does, its behavior would not appear to be anomalous because those thresholds must allow it as normal behavior to avoid constantly flagging the email server as infected.

There are many ways to avoid this problem. One way would be to manually maintain a list of servers (or more generally, machines with unusual profiles), but manually maintaining such a list is error prone and labor intensive. A better way must exist.

In fact there has been substantial research effort in clustering entities based on their expressed characteristics in both the realm of statistics and computer-science. We have leveraged this body of work and applied existing tools to automatically categorize the hosts in a network such that different thresholds can be applied to different groups of hosts.

We examined a range of clustering techniques, behavior characteristics

to cluster against, and number of clusters to create. We have found that using k-means clustering to separate the hosts into just two groups allows us to improve overall performance. We cluster based on only a single feature of the hosts, the number of destinations contacted during a training period.

7.4. SWORD2

We have combined the above principles into a new Host Activity-based Worm Catcher, or SWORD2, for short. SWORD2 uses the BDD and QPD detectors outlined above, and declares a host to be infected with a worm when either detection heuristic raises the alarm. SWORD2 observes legitimate network activity for a period of time to cluster hosts into two groups and generate thresholds for each cluster. In Chapter VIII we examine the performance of the SWORD2 detector and show it to perform better than the detectors evaluated in Chapters V and VI.

CHAPTER VIII

AN EVALUATION OF THE SWORD2 DETECTOR

Having described our new detector, SWORD2, in Chapter VII, we now evaluate its ability to detect worms and compare it to the detectors we evaluated in Chapter V. We will first consider the Burst Duration Detector and the Quiescent Period Detector in isolation to determine their performance characteristics. We then add the clustering component to see how it improves their performance before combining the two detectors and evaluating the performance of the SWORD2 detector as a whole.

To evaluate the detectors, we perform the same set of experiments using the same network traces and worm simulations as described in Chapter V. This gives us a fair comparison between these detectors and existing ones, and allows us to objectively assess their performance.

As discussed in Chapter VI, a worm detector is worthless if it can be easily evaded. After considering SWORD2's performance against naive worms, we develop and evasive worm that attempts to avoid exhibiting the behaviors that SWORD2 relies on. We measure this evasive worm's effective connection rate, evasion rate, and maximum effective rate against SWORD2 and show that SWORD2 is more effective at limiting an evasive worm's rate of spread than the other detectors we evaluated.

8.1. Burst Duration Detector

We start with the Burst Duration Detector (BDD for short). This detector relies on detecting anomalous first-contact connection rates. It is configured to use burst sizes of between 10 and 4000. Smaller burst sizes included bursts that happened so quickly in legitimate traffic that they did not help in detection. Larger bursts were also not useful during detection. Either the worm was already caught before 4000 connections had passed or else experiment ended before the worm had scanned 4000 times.

Training

BDD performs it's training like most of the other detectors we have evaluated. It simply examines the traffic from a period of normal operation of the network, and uses this as a baseline for establishing thresholds that mark behavior as anomalous.

For each different burst size to be considered, we find the shortest duration during which that size burst of first-contact connections was generated by a host. A burst multiplier, α is applied to each burst duration to establish a threshold for that burst size.

Table 8.1 shows the burst multipliers that were selected for each environment to set the false positive rate to two. We do not display the individual burst durations as there are nearly 4000 per environment. A surprising development is

TABLE 8.1. **Parameter choices for the Burst Duration Detector.** The parameter choice for each environment is what was chosen to achieve a false positive rate of two falsely identified hosts per hour during the experiment.

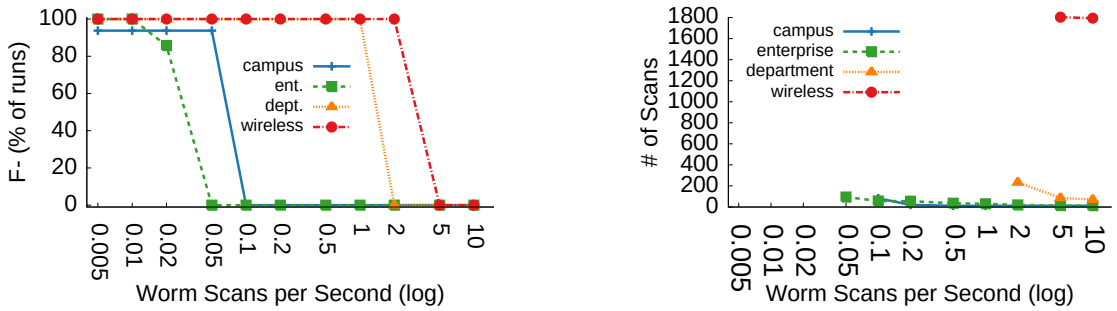
Detector	Parameter	Default	Enterprise	Campus	Dept.	Wireless
BDD	α	0.5	0.74	0.3	1.05	0.24

that for the department environment, the multiplier is actually 1.05. This may seem odd, but recall that the minimum burst durations are determined in the training trace, but the parameters are tuned against the evaluation trace. In this environment, an α of greater than one was required to fix the false positive rate at two.

Without Clustering

The Burst Duration Detector is similar to the RBS and MRW detectors, and we would therefore expect it to perform similarly to those detectors. In Figure 8.1 we show the performance of the BDD detector without the clustering component being used. This means that BDD applies the same thresholds to all hosts, and must establish thresholds that yield a suitable false positive rate when applied to the entire network.

In comparing the false negative results for the BDD detector with no clustering (Figure 8.1a) against the results for the MRW and RBS detectors (Figure 5.2d and Figure 5.2c respectively) we can see that in fact the BDD detector performs quite well. BDD shows more sensitivity than RBS in all but the wireless environment, where it performed equivalently. It outperformed MRW in the



(a) False Negatives

(b) Detection Latency

FIGURE 8.1. **F- and detection latency for BDD with no clustering** when running against random scanning worms infecting randomly selected hosts.

campus and enterprise environments and equivalently in the wireless environment, but performed worse in the department environment. In general, the sensitivity of the BDD detector compares well with any of the detectors evaluated in the detector comparison in Chapter V.

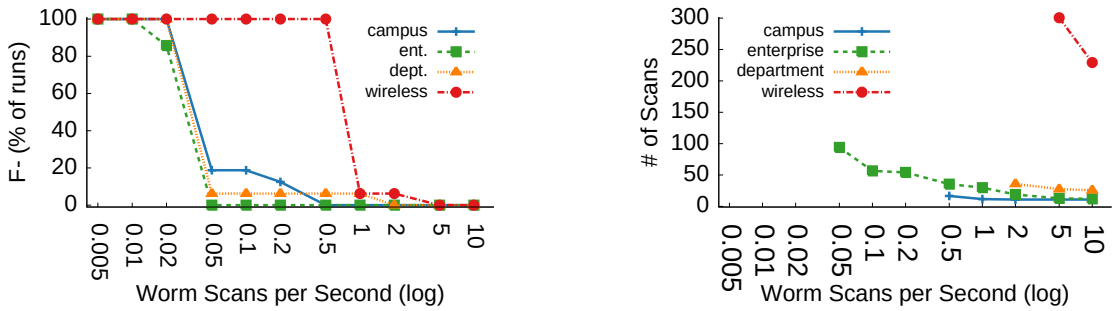
The latency of the BDD (Figure 8.1b) detector without clustering also outperforms the MRW and RBS detectors (Figure 5.3d and Figure 5.3c). At a scan rate of 0.2 connections per second, the BDD detector has an average latency of 19.25 connections in the campus environment compared to 47.33 connections for MRW. A similar pattern holds for the campus environment, only in the department environment does MRW significantly outperform BDD. For high speed worms (greater than or equal to 5 connections per second), RBS shows a significantly lower latency than BDD. RBS' sensitivity is so poor, however, that we don't have any results to compare against slower worms.

With Clustering

When we add the clustering component to BDD, we can see that clustering makes a significant difference in the overall performance of the detector (Figure 8.2). Specifically, the clustering makes the most impact in the environments that showed the worst performance. In the department environment, BDD was sensitive to worms only down to 2 connections per second without clustering. With clustering, however, it is sensitive to worm traffic all the way down to 0.05 connections per second. Similarly, for the wireless environment BDD becomes sensitive to worm traffic down to 1 connection per second when it was previously only sensitive at 5 connections per second.

The latency shows a good improvement in the problem environments, but a less noticeable improvement in the other environments. In the department environment, latency in detecting a worm scanning at 2 scans per second drops from an average of 233.37 to an average of 35.81 connections. A similarly dramatic improvement is seen in the wireless environment, where the latency at 5 scans per second drops from an average of 1803.94 connections to an average of 300.62 connections.

This dramatic improvement is seen because the range in activity levels between the busiest hosts and least busy hosts is greatest in the wireless and department environments. The campus and enterprise environments are more similar from host to host and so see a less noticeable improvement.



(a) False Negatives

(b) Detection Latency

FIGURE 8.2. **F- and detection latency for BDD with clustering** when running against random scanning worms infecting randomly selected hosts.

8.2. Quiescent Period Detector

The Quiescent Period Detector (or QPD for short) detects worms by measuring the duration of their active periods. An active period is defined as the duration of a period during which first-contact connections happen with no more than the specified quiescent period between them. QPD uses a series of different quiescent periods, each with their own threshold. The violation of any of the quiescent period threshold raises the alarm.

Training

QPD is trained by observing legitimate traffic and measuring the active periods. It is configured to use quiescent periods of: 30, 60, 90, 120, 150, 180, 210, and 240 seconds.

For each quiescent period duration, it measures the durations of all of the active periods that are bounded by a quiescent duration of at least that length.

The mean and standard deviations of all the measured active durations are calculated. The threshold for the quiescent period size is the mean plus α times the standard deviation. Alpha can be tweaked for different environments to fix the false positives at a specific value. Table 8.2 shows the standard deviation multiplier selected in each environment to fix the false positives at two per hour.

Without clustering

The QPD detector does not have any clear parallels to a detector evaluated in Chapter V, so we will examine it the context of all the detectors. QPD shows very promising sensitivity (see Figure 8.3a). It outperforms all the detectors except TRW in the wireless environment where it detects the worm 100% of the time at just 0.1 scans per second. In the other environments, it detects the worm at 0.05 scans per second. This performance is marginally worse than TRW (which detects the worm down to 0.02 scans per second), but is better than TRWRBS, MRW, and RBS; and beats DSC and PGD in all environments except for the enterprise environment.

QPDs excellent sensitivity is coupled with good, but not great, detection latency. Because the QPD detector waits for expected quiescent periods that do not come, it is limited by clock time in its worm detection. This means that the rate of worm scanning has very little impact on how fast (in clock time) the worm is discovered, which means that faster scanning worms will achieve more scans before detection than slow scanning worms will. With the parameters used, the

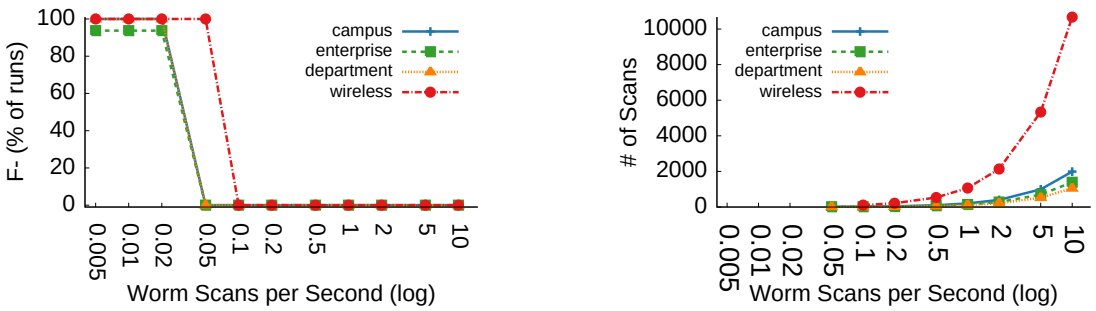
TABLE 8.2. **Parameter choices for QPD.** The parameter choice for each environment is what was chosen to achieve a false positive rate of two falsely identified hosts per hour during the experiment.

Detector	Parameter	Default	Enterprise	Campus	Dept.	Wireless
QPD	α	3.5	2.3	8.0	4.4	3.4

worm is detected in an average of less than 22 connections at 0.1 scans per second in all environments except for the wireless environment where it takes an average of nearly 110 scans to detect the worm. This detection latency is substantially better than MRW, PGD, and TRWRBS at the same worm scanning rates. RBS and DSC weren't able to detect the worm at that scanning rate so can't be compared directly. TRW is substantially faster in the enterprise and campus environment, slightly slower in the department environment, and has about half the latency in the wireless environment. These latency numbers are very encouraging for the QPD detector. These, however, also also the very best latency numbers that are reported for QPD. As the worm scanning rate goes up, so does the latency in QPD detecting the worm. At 10 scans per second, QPD's detection latency jumps to over 1000 scans in the campus, enterprise, and department environments, and to more than 10,000 in the campus environment! These poor latency results are one reason that the QPD detector is not worthwhile on its own.

With clustering

Figure 8.4 reports QPD's false negative and latency results when clustering is used to establish better thresholds. Clustering improves QPD's sensitivity, at the



(a) False Negatives

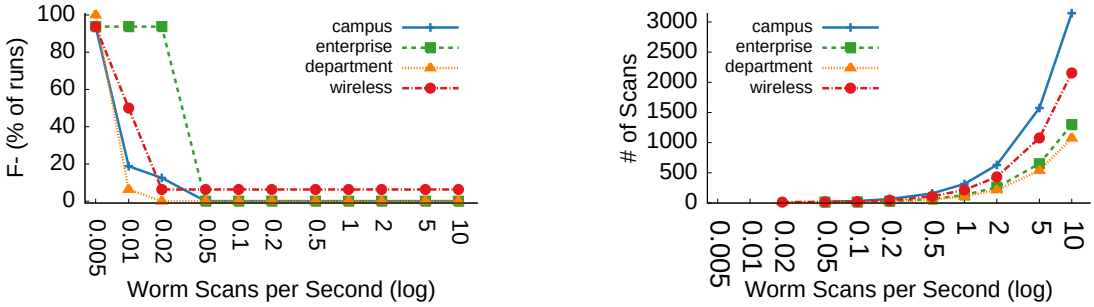
(b) Detection Latency

FIGURE 8.3. **F- and detection latency for QPD with no clustering** when running against random scanning worms infecting randomly selected hosts.

cost of a single infection in the wireless environment that is no longer detected. In the department environment, 100% of the infections are detected at 0.02 scans per second, whereas none of them were detected in the unclustered experiment. False negatives were less than 20% in the campus environment where they were 100% in the unclustered environment.

The latency results showed a less noticeable improvement with the exception of the wireless environment. Here, the detection latency at 0.1 scans per second dropped from just under 110 to less than 24, a more than four-fold improvement. The improvement applied at all scanning rates, reducing the average detection latency to 2155 scans at 10 scans per second. This performance is still too poor to deploy this detector on its own, but it is a significant improvement.

The high sensitivity of the QPD detector makes it an excellent scheme for detecting slow scanning worms. Its poor latency for faster scanning worms means that it needs to be coupled with another detector to handle those scenarios better.



(a) False Negatives

(b) Detection Latency

FIGURE 8.4. **F- and detection latency for QPD with clustering** when running against random scanning worms infecting randomly selected hosts.

8.3. SWORD2 Detector Compared with Existing Works

The SWORD2 detector is a simple combination of the BDD and QPD detection mechanisms. SWORD2 raises an alarm when either BDD or QPD is tripped. The thresholds must be modified for the detectors when operating together like this to achieve the desired false positive rate, so the results are not a simple combination of the above results.

Training

The training for SWORD2 is the same as the training for the two component detectors. A period of legitimate network activity is observed, and the thresholds are calculated from it. SWORD2 uses the same limits on burst sizes used in the BDD section above, and the same quiescent period durations defined above. Table 8.3 shows the parameter values used by SWORD2 to fix the false positives at two. Having established both that the component mechanisms are effective without

TABLE 8.3. **Parameter choices for SWORD2.** The parameter choices for each environment are what were chosen to achieve a false positive rate of two falsely identified hosts per hour during the experiment.

Detector	Parameter	Default	Enterprise	Campus	Dept.	Wireless
BDD	α	0.5	0.66	0.22	1.05	0.24
QPD	α	3.5	3.6	8.0	4.4	5.6

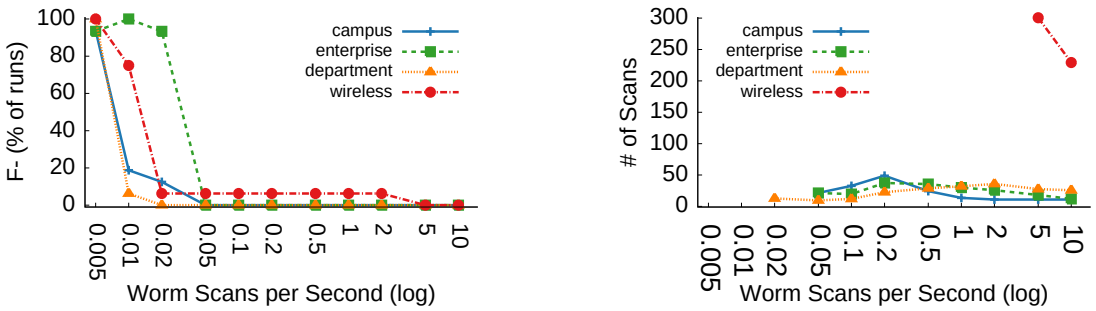
clustering and that clustering does improve their performance, we see no reason to present the SWORD2 results without clustering.

Results

Figure 8.5a shows the false negative rate achieved by the SWORD2 detector against a random scanning worm. At the slower scanning rates the false negatives look quite similar what was shown by the QPD detector. The worm was detected at a scanning rate of 0.05 connections per second in every scenario except for a single host in the wireless environment.

The latency graph shows much more interesting results (Figure 8.5b). Unlike the QPD detector, with the SWORD2 detector the detection latency is low at all scanning rates. In the campus, enterprise, and department traces the average detection latency is under 40 scans for all worm scanning rates but one, where the average detection latency is under 50 scans.

We only report detection latency for combinations of environment and scanning rate where the worm was detected in 100% of the experiments. This means that we do not see much of the latency performance in the wireless



(a) False Negatives

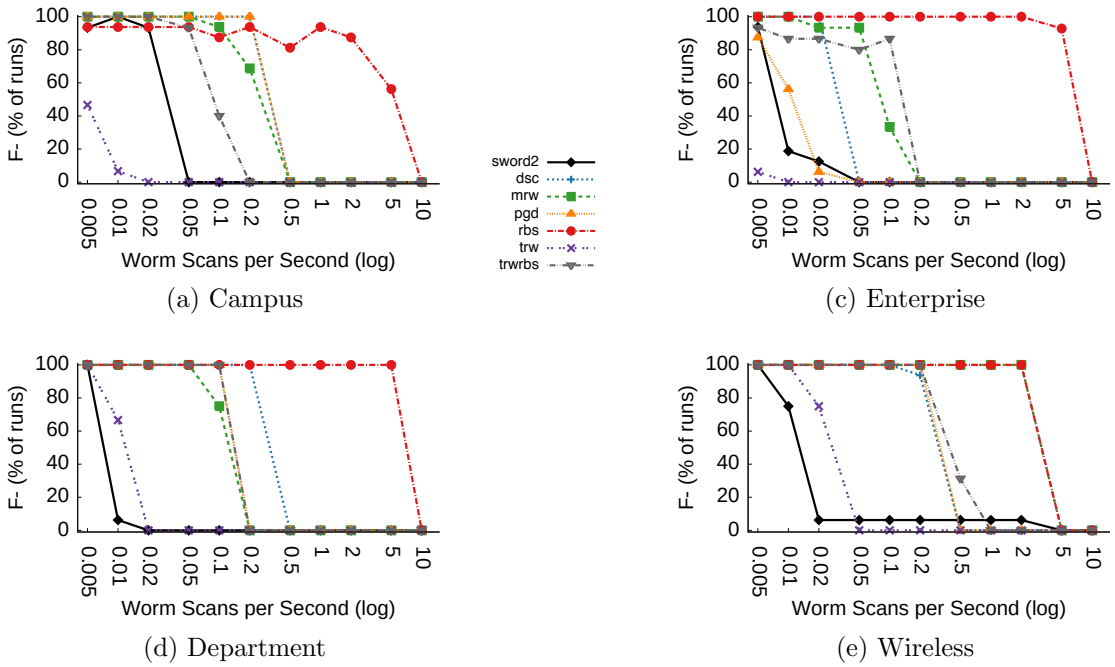
(b) Detection Latency

FIGURE 8.5. **F- and detection latency for SWORD2** when running against random scanning worms infecting randomly selected hosts.

environment. However, if we were to relax our restriction and show the detection latency for those 15 scenarios at each scanning rate where the worm was detected in the wireless environment, we would see that the latency is under 67 for all scan rates under 0.2, and under 327 for all worm scan rates. These latency numbers are excellent, and compare well with the detectors we evaluated in Chapter V.

To make a direct head-to-head comparison between SWORD2 and the other detectors easier, Figure 8.6 plots SWORD2 and the other detectors all on the same graph.

In the campus environment (Figure 8.6a), we can see that the TRW detector is able to detect some worms at slightly slower scan rates than SWORD2. It is the only detector to do so, however, and does not detect 100% of the infections at any scan rate slower than SWORD2 does. The enterprise environment (Figure 8.6c) shows similar results. Again with TRW showing slightly better sensitivity and this time PGD just barely beating SWORD2 at 0.02 scans per second. The other two



(a) Campus

(c) Enterprise

(d) Department

(e) Wireless

FIGURE 8.6. **False Negatives for all detectors** when running against random scanning worms infecting randomly selected hosts.

environments, however, (Figure 8.6d and Figure 8.6e) show SWORD2 with the best sensitivity, detecting worm infections at slower scanning rates than any other detector.

A similar set of graphs for the detection latency is difficult make legible because of the wide ranging latency values and because no data is present for so many of the points. Instead, we present a simplified view, with the average taken across all scanning rates where the worm was detected for each detector and environment. These results are presented in Table 8.4.

The only two detectors that beat the SWORD2 detector in the sensitivity tests were the PGD detector (in one environment only) and the TRW detector

TABLE 8.4. **Average detection latency for all detectors.** Each value represents the average detection latency across all scan rates for a given detector and environment, only including values where all worms were detected in a given environment.

Detector	Campus	Enterprise	Department	Wireless
SWORD2	21.73	24.97	22.99	264.94
DSC	2.00	22.00	19.00	15.93
MRW	28.88	51.70	43.64	1014.16
PGD	93.80	28.11	25.81	621.75
RBS	17.36	4.25	26.44	349.53
TRW	4.23	11.13	24.75	49.93
TRWRBS	57.97	30.39	58.66	167.95

(in two environments). Against all other detectors SWORD2 has either better sensitivity or detection latency, and in many cases both.

SWORD2 has a lower average detection latency than PGD in all environments here, including a latency of less than half in the wireless environment. The TRW detector had better sensitivity in two environments and has a lower detection latency in three of the four environments. These tests make the TRW detector appear to be a better detector than the SWORD2 detector. However, as we saw in Chapter VI, a clever worm can evade the TRW detector by employing known neighbors to befuddle the detector. In the next section we show that the SWORD2 detector is dramatically better once evasive worms are taken into account.

8.4. SWORD2 vs Evasive Worms

A good worm detector must be effective not only against naive worms, but

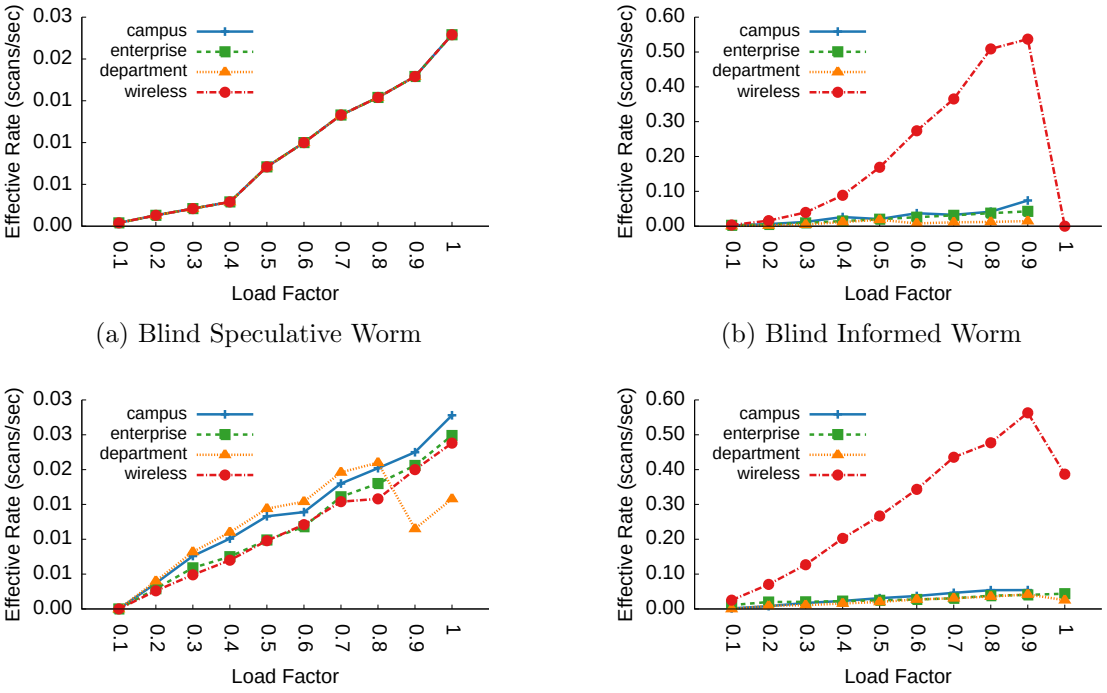
also against worms that are actively trying to evade detection. We now evaluate SWORD2 against worms that are actively trying to evade detection by it. We employ the same methodology and metrics that were used in Chapter VI. We test SWORD2 against *blind/perceptive* and *speculative/informed* worms and measure the worms effective scan rate and evasion rate. We then measure the maximum effective scan rate achieved with a less than 10% chance of detection.

To evade detection by the SWORD2 detector, a worm must ensure that it has sufficient quiescent periods to evade QPD, while also limiting its bursts of connections to avoid triggering BDD. The combination of these two heuristics puts significant constraints on the ability of the worm to scan.

Listing 6.1 shows the core logic of the blind version of this evasive worm. The worm runs internal versions of both the QPD and BDD detectors. It first checks to see whether a scan will violate any of the QPD constraints. If it will, the worm waits long enough to end the current active period for the QPD constraint in question. After eliminating QPD as a constraint the BDD durations are checked to ensure that the BDD detector won't be triggered.

The perceptive version of the worm is similar (see Listing 6.2). It uses a more sophisticated waiting algorithm against the QPD detector to account for legitimate activity.

The blind speculative version of the worm cannot achieve an effective scan rate of greater than 0.03 scans per second in any scenario (see Figure 8.7a). In



(a) Blind Speculative Worm

(b) Blind Informed Worm

(c) Perceptive Speculative Worm

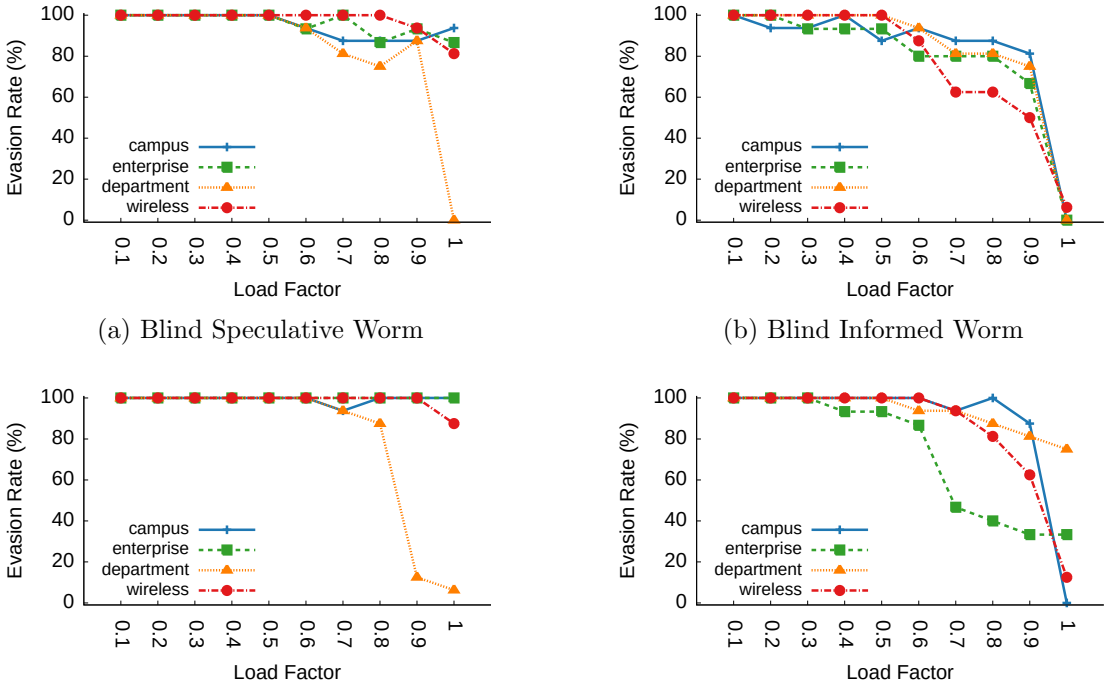
(d) Perceptive Informed Worm

FIGURE 8.7. **Effective scanning rate vs SWORD2** as a function of Load Factor

the department environment, even this low scan rate still gives an evasion rate of 0% when the load factor is 1 in the department environment (see Figure 8.7a). The perceptive speculative version of the worm does not improve the effective rate at all (Figure 8.7c), but does improve the evasion rate in all but the department environment (Figure 8.8c).

The informed versions of the worm do much better in the wireless environment, with the worm able to achieve an effective rate nearly 10x greater than the speculative worm was able to (Figure 8.7b and Figure 8.7d).

The informed versions of the worm do much better in the wireless environment, with the worm able to achieve an effective rate nearly 10x greater



(a) Blind Speculative Worm

(b) Blind Informed Worm

(c) Perceptive Speculative Worm

(d) Perceptive Informed Worm

FIGURE 8.8. **Evasion rate vs SWORD2** as a function of Load Factor

than the speculative worm was able to (Figure 8.7b and Figure 8.7d).

The best evaluation of the detector is the maximum effective rate achieved by the evasive worm while running less than a 10% chance of being detected. We plot the maximum effective rate achieved by the evasive worms against their respective detectors.

In the enterprise environment (Figure 8.9), the maximum effective rate allowed by the SWORD2 detector to any variety of evasive worm was 0.02 scans per second. The PGD detector is the only one to beat SWORD2, the other detectors all perform significantly worse. PGD detected the blind evasive worm varieties in every scenario in this environment because of the window that had

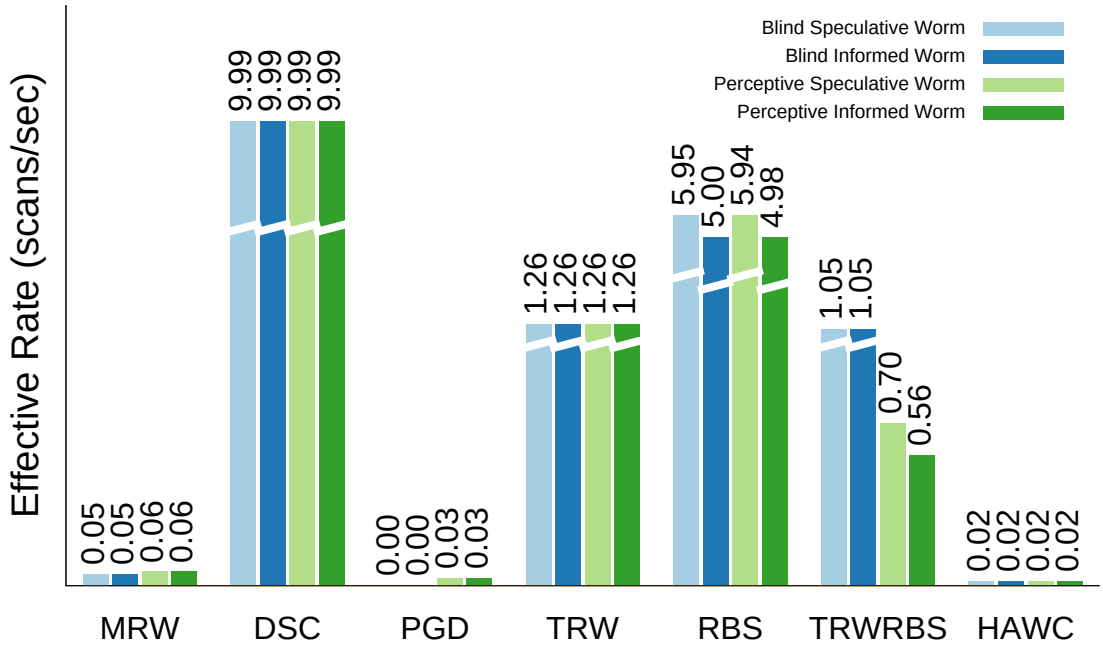


FIGURE 8.9. **Maximum effective rate in enterprise environment**

a threshold that was only one scan over the legitimate traffic for this period. However, the perceptive variant of the worm was able to achieve a maximum effective rate of 50% greater against the PGD detector than against the SWORD2 detector.

For the campus environment (Figure 8.10), the benefits of the SWORD2 detector were more pronounced, beating all other detectors by at least a factor of 2. In this environment, no other detector came close to limiting the maximum effective rate of the evasive worms as well as SWORD2 did.

Figure 8.11 shows similar results for the department environment. SWORD2 outperforms all other detectors by at least a factor of three for all evasive worm varieties.

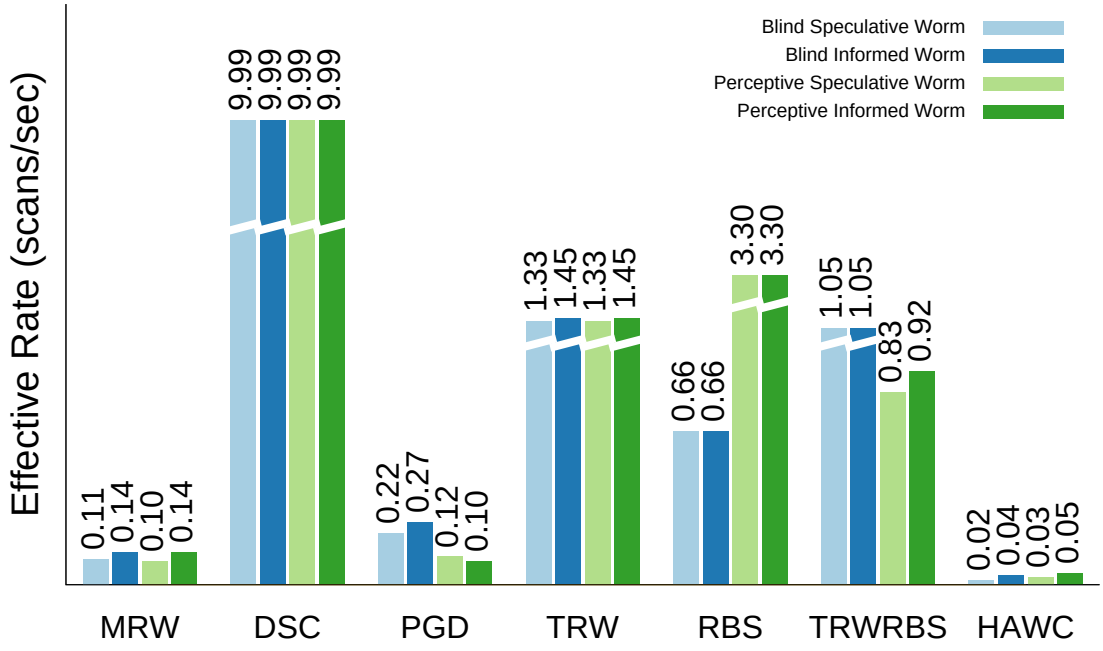


FIGURE 8.10. **Maximum effective rate in campus environment**

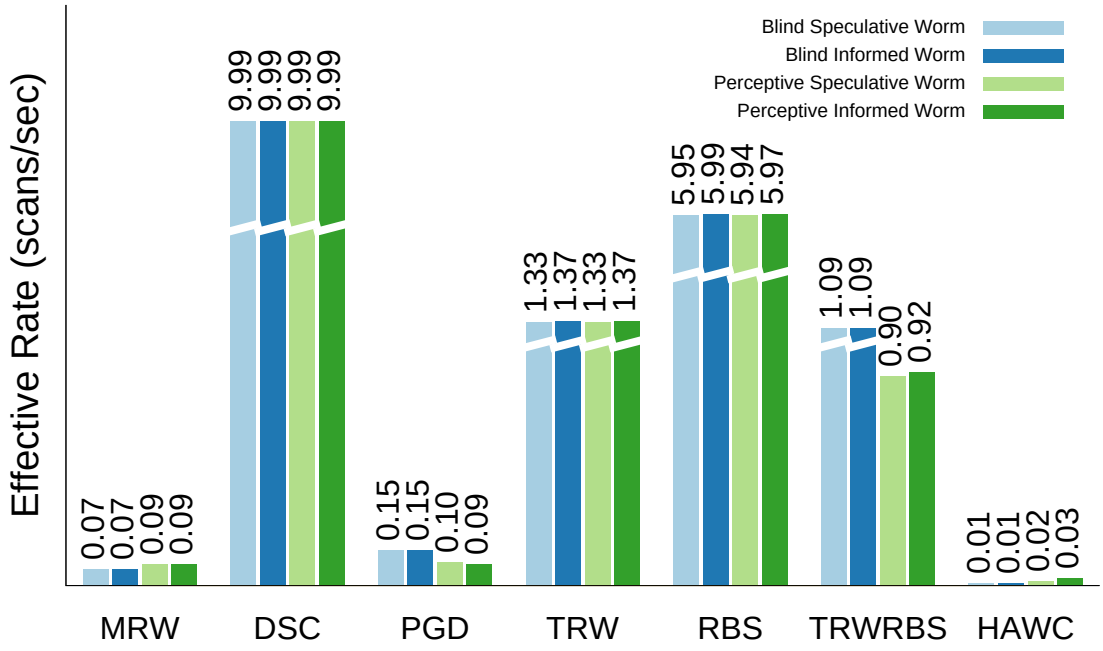


FIGURE 8.11. **Maximum effective rate in department environment**

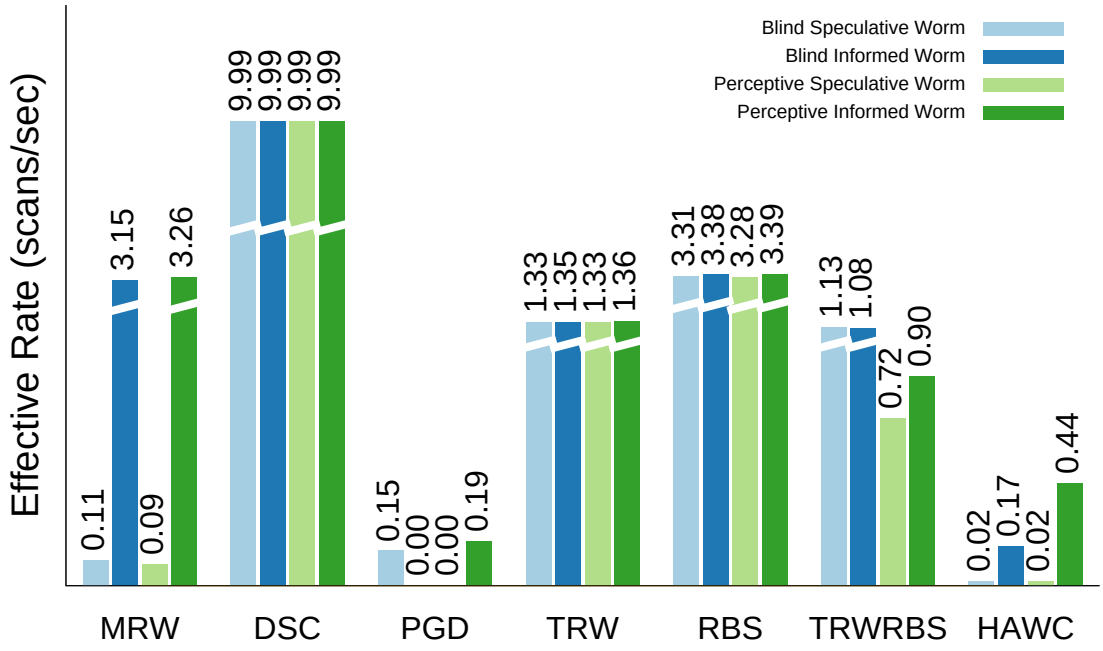


FIGURE 8.12. **Maximum effective rate in wireless environment**

Finally, Figure 8.12 shows that again, the PGD detector was able to outperform SWORD2 in some scenarios. But even here, SWORD2 outperforms PGD by a factor of 7 against blind speculative worms.

8.5. Analysis

In this chapter we have shown that the SWORD2 detector significantly outperforms all other detectors. The TRW detector does perform slightly better against naive worms, but against evasive worms was outperformed by a factor of 60 in most environments against most evasive worm types. TRWs best performance against evasive worms was coming within a factor of 3 against a perceptive informed worm in the wireless environment. This significant superiority against

evasive worms offsets any minor advantage TRW had over SWORD2 against naive worms.

The PGD detector does outperform SWORD2 in 5 of the 16 evasive worm scenarios (four evasive worm types by four environments), but is dominated in the remaining 11 scenarios. It is also outperformed by SWORD2 against naive worms.

The only other detector to come close is the MRW detector, which is consistently outperformed by SWORD2 against naive worms. MRW's best performance against evasive worms is to get within a factor of two of the SWORD2 detector. It is soundly beaten in every scenario.

None of the other detectors present even an appreciable level of competition.

CHAPTER IX

CONCLUSION

In this work we have presented an introduction to network worms, including information about the way they propagate and case studies of real worms. This information highlights the risks that worms present to our increasingly networked world. We then presented a taxonomy and survey of existing worm detection mechanisms. There are a wide variety of detection mechanisms spanning a number of classes of detection including content-, behavior-, host-, and honeypot-based algorithms. We chose to focus on behavior-based worm detection because of the ease of deployment and attack agnostic attributes they possess. We selected the six most promising behavior-based detection systems and developed a framework to enable easy evaluation of them. This framework enables us to fairly test the detectors across a range of scenarios.

The evaluation revealed that no detector stands out as being a panacea. The TRW detector arguably performed the best in the evaluation, but a shortcoming was observed against topological based worms. These worms do not exhibit the same connection failure behavior that random scanning worms do, and so are not detected as easily by TRW.

Chapter VI took our evaluation one step further and considered worms that deliberately attempt to evade detection by a specific detector. In this chapter we

introduced a range of evasive worm *capabilities* and measured the evasive worms evasion rate and effective scan rate. Against evasive worms, the detectors based on *first-contact connection* rates outperformed other detectors because this behavior is an essential behavior of worms, it cannot be masked or avoided. The PGD and MRW worms showed the best ability to prevent an evasive worm from scanning quickly.

Taking the information we learned from the performance of the evasive worms, we established principles for a new detector that should show better performance. In Chapter VIII we evaluate the performance of this new SWORD2 detector and show that it outperforms existing detectors against both naive and evasive worms.

9.1. Contributions

The primary contributions of this work are (1) the framework for evaluating behavior-based worm detectors (2) the comparison of existing behavior-based detectors (3) the evaluation of evasive worms (4) a new worm detector that outperforms existing behavior-based worm detectors.

In existing publications, worm detection systems are evaluated against a wide range of network traffic and worms. A framework that makes it easy to compare a detector against other detectors using standard network traces and worm implementations will allow researchers to more quickly examine the performance of their ideas and to present them with better evaluation results that are easily

comparable with other detectors.

The comparison of existing detectors using this framework provides a valuable benchmark for the worm detection community. Previous to the publication of this work, it was difficult to directly compare detectors. Some detectors, such as the RBS detector appeared to perform very well in the experiments in their original publication. Showing their performance with standardized traces and metrics reveals some shortcomings in their implementation, giving researchers a more complete picture of their overall value.

Furthermore, detectors should be evaluated not only against naive worms, but also against worms that deliberately try to evade the detector. It is shortsighted to assume that worm authors will not choose to attempt to avoid a popular detector deployed against them. Detectors that seem relatively effective when deployed against naive worms, such as DSC, are shown to be nearly worthless against worms that are attempting to avoid exhibiting the behavior trait the detector relies on. This evaluation is a significant step forward in the worm detection research field.

Finally, the SWORD2 detector represents a substantial improvement in the performance of behavior-based worm detection. It's performance against naive worms was better than all detectors except for TRW which it performed nearly as well as; and it substantially outperformed (by a factor of 60 in most cases) the TRW detector against evasive worms. The only detector to provide somewhat comparable performance against evasive worms was the PGD detector, which

was still outperformed in 11 of the sixteen scenarios. Against the most advanced evasive worm, the *perceptive informed* evasive worm, the SWORD2 detector outperformed PGD by factors of 1.5, 2, and 3 in three of the environments, while being outperformed by PGD by a factor of 2.5 in only one environment. The SWORD2 detector was substantially better than all other detectors against all other evasive worm types.

9.2. Future Work

Despite these contributions, there is still a tremendous opportunity for future research in the field of worm detection. We present the highlights of these topics here, divided into three sections: (1) refining SWORD2 (2) evaluate the impact of different environments (3) training detectors in an adversarial environment (4) handling the evolution of network traffic profiles

Refining and Better Understanding the SWORD2 Detector

This work presents an initial evaluation of the SWORD2 detector, but there is clearly a large body of work that remains to be done on it. The first area of research is a better examination of the way SWORD2 clusters hosts before establishing thresholds. We examined a number of behavioral traits to cluster on and a range of numbers of clusters to produce; but we by no means completed a comprehensive evaluation of this space. We do not know the minimum amount of training time needed to classify a host, nor whether there may be additional

behavior characteristics that provide better clustering results. Further work in this area has the potential to dramatically improve SWORD2's performance.

SWORD2 has been evaluated against small networks, the environment for which it was designed. However, it would be worthwhile to study its performance in much larger networks. Could a large university deploy a single instance of SWORD2 at its gateway to monitor all traffic in the network? SWORD2 may actually perform well in such a scenario, particularly if the clustering performance is improved. We have not evaluated SWORD2 against large networks so do not have a good understanding of how it would perform. It may be that there are tweaks that would be required for SWORD2 to perform well in a large environment.

Additionally, the training mechanisms for SWORD2 should be examined further. We currently use very simplistic methods for obtaining thresholds: the minimum observed burst size and the mean plus standard deviation of the active periods. It is possible that a more sophisticated training mechanism would be able to establish more refined thresholds that would lead to fewer false positives and lower detection latencies.

Different Environments

We have done our evaluations against the network traces available to us, and have seen that the environment significantly impacts the performance of the different detectors. There is a tremendous amount of room for further study here,

however. For example, by creating artificial traces made up of randomly selected hosts from all of the environments, we could attempt to establish some confidence level for how representative these traces are. Will hosts randomly selected from environments with known performance characteristics yield a resulting trace with similar or different characteristics? If we increased the size of the network does it change the overall performance of the detectors?

Training in an Adversarial Environment

The biggest shortcoming of all of the detectors examined here is that they require some form of training on what legitimate traffic looks like in the environment they are deployed in. This is a problem if the network is already infected with a worm, because the training data will be skewed by the presence of the worm traffic. In fact, in most scenarios the worm traffic will be considered normal and the thresholds set such that it is scored that way. This is a huge problem that faces not only the SWORD2 detector, but all behavior-based worm detectors.

It is a general problem of training in an adversarial environment. If you don't know that the corpus you are training on is legitimate, how can you use it for training?

The TRW, TRWRBS, and RBS detectors use an ongoing rolling training period to attempt to mitigate this problem, but that does not solve it completely. A worm can influence the thresholds by gradually increasing its scanning rate.

The Evolution of Network Traffic

Network traffic is constantly changing as new protocols and applications become popular. How does a worm detector adapt to these changes. This work is directly related to the above point about training. Behavior-based worm detectors are application and protocol agnostic, but they still rely on identifying anomalous behavior. As applications come and go, behavior patterns change to suit the application. It is unclear how a detector can adapt along with the changes.

9.3. Conclusion

In this work we have presented the risk that network worms pose to the Internet and have made four contributions toward reducing that risk. Our evaluation framework makes it easier for worm detector authors to quickly evaluate the performance of their new detector. Our comparison of existing detectors establishes a baseline for what a new detector needs to beat to be useful. Our examination of evasive worms provides further insight into the effectiveness of various detection mechanisms and highlights those that can be easily evaded. Finally, our SWORD2 detector outperforms existing detectors, advancing the state of the art of worm detection.

REFERENCES CITED

[1] A. Rubin, "There are now over 700,000 android devices activated every day," twitter, December 2011. [Online]. Available: https://twitter.com/#!/Arubin/status/149329329237667844

[2] ——, "There were 3.7m android devices activated on 12/24 and 12/25," twitter, December 2011. [Online]. Available: https://twitter.com/#!/Arubin/status/151918325260226561

[3] R. Baldwin, "Nearly 1 million iphone 4s and ipad 2s jailbroken in three days," January 2012. [Online]. Available: http://gizmodo.com/5878607/nearly-1-million-iphone-4s-and-ipad-2s-jailbroken-in-three-days

[4] A. Inc., "Over 500,000 apps. for work, play, and everything in between," January 2012. [Online]. Available: http://www.apple.com/iphone/from-the-app-store/

[5] V. Jacobson, D. K. Smetters, J. D. Thornton, M. F. Plass, N. H. Briggs, and R. L. Braynard, "Networking named content," in *Proceedings of the International Conference on Emerging Networking Experiments and Technologies*. New York, NY: ACM Press, December 2009, pp. 1–12.

[6] C. C. Zou, L. Gao, W. Gong, and D. Towsley, "Monitoring and early warning for Internet worms," in *Proceedings of the Conference on Computer and Communications Security*. New York, NY: ACM Press, 2003, pp. 190–199.

[7] J. Wu, S. Vangala, L. Gao, and K. Kwiat, "An effective architecture and algorithm for detecting worms with various scan techniques," in *Proceedings of the Network and Distributed System Security Symposium*. Reston, VA: The Internet Society, 2004.

[8] S. Staniford, V. Paxson, and N. Weaver, "How to 0wn the Internet in your spare time," in *Proceedings of the USENIX Security Symposium*. Berkeley, CA: USENIX, 2002, pp. 149–167.

[9] Z. Chen and C. Ji, "Measuring network-aware worm spreading ability," in *Proceedings of IEEE INFOCOM*. Washington, DC: IEEE Computer Society, 2007.

[10] N. Provos, J. McClain, and K. Wang, "Search worms," in *Proceedings of the Workshop on Rapid Malcode*. New York, NY: ACM Press, 2006, pp. 1–8.

[11] D. Moore, C. Shannon, and K. C. Claffy, "Code-red: A case study on the spread and victims of an Internet worm," in *Proceedings of the ACM Internet Measurement Workshop*. New York, NY: ACM Press, 2002, pp. 273–284.

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

[13] C. Shannon and D. Moore, "The spread of the witty worm," *IEEE Security and Privacy*, vol. 2, no. 4, pp. 46–50, July 2004.

[14] I. Symantec, "The downadup codex," Symantec, Tech. Rep., March 2009. [Online]. Available: http://www.symantec.com/content/en/us/enterprise/media/security_response/whitepapers/the_downadup_codex_ed1.pdf

[15] P. A. Porras, H. Saidi, and V. Yegneswaran, "An analysis of conficker's logic and rendevous points," SRI International, Tech. Rep., March 2009. [Online]. Available: http://mtc.sri.com/Conficker/

[16] C. Economics, "The cost impact of major virus attacks since 1995," Website, February 2004. [Online]. Available: http://www.computereconomics.com/article.cfm?id=936

[17] P. Barford, R. Nowak, R. Willett, and V. Yegneswaran, "Toward a model for sources of Internet background radiation," in *Proceedings of the Passive and Active Measurement Conference*, 2006.

[18] Z. Chen and C. Ji, "Optimal worm-scanning method using vulnerable-host distributions," *International Journal of Security and Networks*, vol. 2, no. 1/2, 2007.

[19] C. C. Zou, D. Towsley, and W. Gong, "On the performance of Internet worm scanning strategies," *Performance Evaluation*, vol. 63, no. 7, pp. 700–723, 2006.

[20] P. A. Porras, H. Saidi, and V. Yegneswaran, "Conficker c analysis," SRI International, Tech. Rep., April 2009. [Online]. Available: http://mtc.sri.com/Conficker/addendumC/

[21] ——, "An analysis of the ikee.b (duh) iPhone botnet," SRI International, Tech. Rep., December 2009. [Online]. Available: http://mtc.sri.com/iPhone/

[22] N. Falliere, L. O Murchu, and E. Chien, "W32.stuxnet dossier," Website, Symantec Corp., Tech. Rep., February 2011.

[23] C. Cowan, C. Pu, D. Maier, J. Walpole, P. Bakke, S. Beattie, A. Grier, P. Wagle, Q. Zhang, and H. Hinton, "Stackguard: Automatic adaptive detection and prevention of buffer-overflow attacks," in *Proceedings of the USENIX Security Symposium*. Berkeley, CA: USENIX, January 1998, pp. 63–78.

[24] J. Condit, M. Harren, S. McPeak, G. C. Necula, and W. Weimer, "CCured in the real world," in *Proceedings of the ACM SIGPLAN Conference on Programming Language Design and Implementation*. New York, NY: ACM Press, 2003, pp. 232–244.

[25] H. Shacham, M. Page, B. Pfaff, E.-J. Goh, N. Modadugu, and D. Boneh, "On the effectiveness of address-space randomization," in *Proceedings of the Conference on Computer and Communications Security*. New York, NY: ACM Press, 2004, pp. 298–307.

[26] S. Antonatos, P. Akritidis, E. P. Markatos, and K. G. Anagnostakis, "Defending against hitlist worms using network address space randomization," in *Proceedings of the Workshop on Rapid Malcode*. New York, NY: ACM Press, 2005.

[27] N. Weaver, S. Staniford, and V. Paxson, "Very fast containment of scanning worms," in *Proceedings of the USENIX Security Symposium*. Berkeley, CA: USENIX, 2004, pp. 29–44.

[28] A. J. Ganesh, D. Gunawardena, P. Key, L. Massouli, and J. Scott, "Efficient quarantining of scanning worms: Optimal detection and coordination," in *Proceedings of IEEE INFOCOM*. Washington, DC: IEEE Computer Society, 2006.

[29] L. Li, P. Liu, Y.-C. Jhi, and G. Kesidis, "Evaluation of collaborative worm containments on DETER testbed," in *Proceedings of the DETER Community Workshop on Cyber Security Experimentation and Test*. Berkeley, CA: USENIX, August 2007.

[30] D. Brumley, L.-H. Liu, P. Poosankam, , and D. Song, "Design space and analysis of worm defense strategies," in *Proceedings of the ACM Symposium on Information, Computer, and Communication Security*. New York, NY: ACM Press, 2006.

[31] T. Liston, "Welcome to my tarpit: The tactical and strategic use of labrea," http://www.threenorth.com/LaBrea/LaBrea.txt, None, Tech. Rep., 2001.

[32] M. Williamson, "Throttling viruses: Restricting propagation to defeat malicious mobile code," in *Proceedings of the Annual Computer Security Applications Conference*. Washington, DC: IEEE Computer Society, 2002.

[33] J. Twycross and M. M. Williamson, "Implementing and testing a virus throttle," in *Proceedings of the USENIX Security Symposium*. Berkeley, CA: USENIX, 2003, pp. 285–294.

[34] M. M. Williamson, "Design, implementation and test of an email virus throttle," in *Proceedings of the Annual Computer Security Applications Conference*. Washington, DC: IEEE Computer Society, December 2003, p. 76.

[35] S. E. Schechter, J. Jung, and A. W. Berger, "Fast detection of scanning worm infections," in *Proceedings of the Symposium on Recent Advances in Intrusion Detection*. Berlin, Heidelberg: Springer-Verlag, 2004.

[36] C. Wong, C. Wang, D. Song, S. Bielski, and G. Ganger, "Dynamic quarantine of Internet worms," in *Proceedings of the International Conference on Dependable Systems and Networks*. Washington, DC: IEEE Computer Society, 2004.

[37] G. Gu, M. Sharif, X. Qin, D. Dagon, W. Lee, and G. Riley, "Worm detection, early warning and response based on local victim information," in *Proceedings of the Annual Computer Security Applications Conference*. Washington, DC: IEEE Computer Society, 2004.

[38] J. Kannan, L. Subramanian, I. Stoica, and R. Katz, "Analyzing cooperative containment of fast scanning worms," in *Proceedings of the USENIX Steps to Reducing Unwanted Traffic on the Internet Workshop (SRUTI)*. Berkeley, CA: USENIX, 2005.

[39] D. Moore, C. Shannon, G. M. Voelker, and S. Savage, "Internet quarantine: Requirements for containing self-propagating code," in *Proceedings of IEEE INFOCOM*, vol. 3. Washington, DC: IEEE Computer Society, 2003, pp. 1901–1910.

[40] N. Weaver and V. Paxson, "A worst-case worm," in *Proceedings of the Workshop on Economics and Information Security*, 2004.

[41] S. Staniford, D. Moore, V. Paxson, and N. Weaver, "The top speed of flash worms," in *Proceedings of the Workshop on Rapid Malcode*. New York, NY: ACM Press, 2004.

[42] Y. Song, M. E. Locasto, A. Stavrou, A. D. Keromytis, and S. J. Stolfo, "On the infeasibility of modeling polymorphic shellcode," in *Proceedings of the Conference on Computer and Communications Security*. New York, NY: ACM Press, October 2007, pp. 541–551.

[43] M. V. Gundy, D. Balzarotti, and G. Vigna, "Catch me, if you can: Evading network signatures with web-based polymorphic worms," in *Proceedings of the USENIX Workshop on Offensive Technologies*. Berkeley, CA: USENIX, August 2007, pp. 1–9.

[44] P. Fogla, M. Sharif, R. Perdisci, O. Kolesnikov, and W. Lee, "Polymorphic blending attacks," in *Proceedings of the USENIX Security Symposium*. Berkeley, CA: USENIX, 2006, pp. 241–256.

[45] M. A. Rajab, F. Monrose, and A. Terzis, "Fast and evasive attacks: Highlighting the challenges ahead," in *Proceedings of the Symposium on Recent Advances in Intrusion Detection*. Berlin, Heidelberg: Springer-Verlag, 2006.

[46] S. P. Chung and A. K. Mok, "Allergy attack against automatic signature generation," in *Proceedings of the Symposium on Recent Advances in Intrusion Detection*. Berlin, Heidelberg: Springer-Verlag, 2006.

[47] R. Perdisci, D. Dagon, W. Lee, P. Fogla, and M. Sharif, "Misleading worm signature generators using deliberate noise injection," in *Proceedings of the IEEE Symposium on Security and Privacy*. Washington, DC: IEEE Computer Society, 2006.

[48] J. Newsome, B. Karp, and D. Song, "Paragraph: Thwarting signature learning by training maliciously," in *Proceedings of the Symposium on Recent Advances in Intrusion Detection*. Berlin, Heidelberg: Springer-Verlag, 2006.

[49] Z. Liang and R. Sekar, "Fast and automated generation of attack signatures: A basis for building self-protecting servers," in *Proceedings of the Conference on Computer and Communications Security*. New York, NY: ACM Press, 2005.

[50] J. Tucek, J. Newsome, S. Lu, C. Huang, S. Xanthos, D. Brumley, Y. Zhou, and D. Song, "Sweeper: A lightweight end-to-end system for defending against fast worms," in *Proceedings of the EuroSys Conference*, 2007.

[51] D. Dagon, X. Qin, G. Gu, W. Lee, J. Grizzard, J. Levin, and H. Owen, "Honeystat: Local worm detection using honeypots," in *Proceedings of the Symposium on Recent Advances in Intrusion Detection*, ser. Lecture Notes in Computer Science, vol. 3224. Berlin, Heidelberg: Springer-Verlag, September 2004, pp. 39–58.

[52] J. Newsome and D. Song, "Dynamic taint analysis for automatic detection, analysis, and signature generation of exploits on commodity software," in *Proceedings of the Network and Distributed System Security Symposium*. Reston, VA: The Internet Society, February 2005.

[53] J. R. Crandall, Z. Su, S. F. Wu, and F. T. Chong, "On deriving unknown vulnerabilities from zero-day polymorphic and metamorphic worm exploits," in *Proceedings of the Conference on Computer and Communications Security*. New York, NY: ACM Press, 2005.

[54] J. R. Crandall and F. T. Chong, "Minos: Control data attack prevention orthogonal to memory model," in *Proceedings of the ACM/IEEE International Symposium on Microarchitecture*. Washington, DC: IEEE Computer Society, 2004, pp. 221–232.

[55] M. Costa, J. Crowcroft, M. Castro, A. Rowstron, L. Zhou, L. Zhang, and P. Barham, "Vigilante: End-to-end containment of Internet worms," *Operating Systems Review*, vol. 39, no. 5, pp. 133–147, 2005.

[56] D. J. Malan and M. D. Smith, "Host-based detection of worms through peer-to-peer cooperation," in *Proceedings of the Workshop on Rapid Malcode*. New York, NY: ACM Press, 2005.

[57] ——, "Exploiting temporal consistency to reduce false positives in host-based, collaborative detection of worms," in *Proceedings of the Workshop on Rapid Malcode*. New York, NY: ACM Press, 2006, pp. 25–32.

[58] C. Kreibich and J. Crowcroft, "Honeycomb: Creating intrusion detection signatures using honeypots," in *Proceedings of the Workshop on Hot Topics in Networks*. Berkeley, CA: USENIX, 2003, pp. 51–56.

[59] Y. Tang, H. Hu, X. Lu, and J. Wang, "Honids: Enhancing honeypot system with intrusion detection models," in *Proceedings of the IEEE International Information Assurance Workshop*. Washington, DC: IEEE Computer Society, 2006.

[60] Y. Tang and S. Chen, "Defending against Internet worms: A signature-based approach," in *Proceedings of IEEE INFOCOM*. Washington, DC: IEEE Computer Society, 2005.

[61] M. Roesch, "Snort - lightweight intrusion detection for networks," in *Proceedings of the Conference on Systems Administration*. Berkeley, CA: USENIX, November 1999, pp. 229–238.

[62] V. Paxson, "Bro: A system for detecting network intruders in real-time," in *Proceedings of the USENIX Security Symposium*. Berkeley, CA: USENIX, 1998.

[63] B. N. Chun, J. Lee, and H. Weatherspoon, "Netbait: A distributed worm detection service," Intel Research Berkeley, Tech. Rep. IRB-TR-03-033, 2003.

[64] S. Chen and S. Ranka, "Detecting Internet worms at early stage," *Journal on Selected Areas in Communications*, vol. 23, 2005.

[65] H.-A. Kim and B. Karp, "Autograph: Toward automated, distributed worm signature detection," in *Proceedings of the USENIX Security Symposium*. Berkeley, CA: USENIX, August 2004, pp. 271–286.

[66] S. Singh, C. Estan, G. Varghese, and S. Savage, "Automated worm fingerprinting," in *Proceedings of the Symposium on Operating System Design and Implementation*. Berkeley, CA: USENIX, 2004, pp. 45–60.

[67] B. Madhusudan and J. Lockwood, "Design of a system for real-time worm detection," in *Proceedings of the IEEE Symposium on High Performance Interconnects*. Washington, DC: IEEE Computer Society, 2004.

[68] P. Gopalan, K. Jamieson, P. Mavrommatis, and M. Poletto, "Signature metrics for accurate and automated worm detection," in *Proceedings of the Workshop on Rapid Malcode*. New York, NY: ACM Press, 2006, pp. 65–72.

[69] K. Wang, G. Cretu, and S. J. Stolfo, "Anomalous payload-based worm detection and signature generation," in *Proceedings of the Symposium on Recent Advances in Intrusion Detection*. Berlin, Heidelberg: Springer-Verlag, 2005.

[70] K. Wang and S. J. Stolfo, "Anomalous payload-based network intrusion detection," in *Proceedings of the Symposium on Recent Advances in Intrusion Detection*, ser. Lecture Notes in Computer Science, vol. 3224. Berlin, Heidelberg: Springer-Verlag, September 2004, pp. 203–222.

[71] D. Bolzoni, E. Zambon, S. Etalle, and P. Hartel, "Poseidon: A 2-tier anomaly-based network intrusion detection system," in *Proceedings of the IEEE International Information Assurance Workshop*. Washington, DC: IEEE Computer Society, 2006.

[72] K. Wang, J. J. Parekh, and S. J. Stolfo, "Anagram: A content anomaly detector resistant to mimicry attack," in *Proceedings of the Symposium on Recent Advances in Intrusion Detection*. Berlin, Heidelberg: Springer-Verlag, 2006.

[73] S. P. Chung and A. K. Mok, "Advanced allergy attacks: Does a corpus really help?" in *Proceedings of the Symposium on Recent Advances in Intrusion Detection*, ser. Lecture Notes in Computer Science, vol. 4637. Berlin, Heidelberg: Springer-Verlag, September 2007, pp. 236–255.

[74] J. Newsome, B. Karp, and D. Song, "Polygraph: Automatically generating signatures for polymorphic worms," in *Proceedings of the IEEE Symposium on Security and Privacy*. Washington, DC: IEEE Computer Society, 2005.

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

[76] Z. Li, L. Wang, Y. Chen, and Z. Fu, "Network-based and attack-resilient length signature generation for zero-day polymorphic worms," in *Proceedings of the IEEE International Conference on Network Protocols*. Washington, DC: IEEE Computer Society, October 2007, pp. 164–173.

[77] V. Berk, G. Bakos, and R. Morris, "Designing a framework for active worm detection on global networks," in *Proceedings of the IEEE International Information Assurance Workshop*. Washington, DC: IEEE Computer Society, 2003.

[78] S. G. Cheetancheri, J. M. Agosta, D. H. Dash, K. N. Levitt, J. Rowe, and E. M. Schooler, "A distributed host-based worm detection system," in *Proceedings of the SIGCOMM Workshop on Large-Scale Attack Defense*. New York, NY: ACM Press, 2006, pp. 107–113.

[79] T. Bu, A. Chen, S. V. Wiel, and T. Woo, "Design and evaluation of a fast and robust worm detection algorithm," in *Proceedings of IEEE INFOCOM*. Washington, DC: IEEE Computer Society, 2006.

[80] V. Yegneswaran, P. Barford, and S. Jha, "On the design and utility of Internet sinks for network abuse monitoring," in *Proceedings of the Symposium on Recent Advances in Intrusion Detection*. Berlin, Heidelberg: Springer-Verlag, 2004.

[81] M. A. Rajab, F. Monrose, and A. Terzis, "On the effectiveness of distributed worm monitoring," in *Proceedings of the USENIX Security Symposium*. Berkeley, CA: USENIX, 2005.

[82] S. Stafford, J. Li, and T. Ehrenkranz, "On the performance of SWORD in detecting zero-day-worm-infected hosts," in *Proceedings of the Symposium on Performance Evaluation of Computer and Telecommunication Systems*, vol. 38.3, July 2006, pp. 559 – 566.

[83] V. Sekar, Y. Xie, M. K. Reiter, and H. Zhang, "A multi-resolution approach for worm detection and containment," in *Proceedings of the International Conference on Dependable Systems and Networks*. Washington, DC: IEEE Computer Society, 2006.

[84] J. Jung, V. Paxson, A. W. Berger, and H. Balakrishnan, "Fast portscan detection using sequential hypothesis testing," in *Proceedings of the IEEE Symposium on Security and Privacy*. Washington, DC: IEEE Computer Society, 2004.

[85] J. Jung, R. Milito, and V. Paxson, "On the adaptive real-time detection of fast-propagating network worms," *Journal on Computer Virology*, vol. 4, no. 1, pp. 197–210, February 2008.

[86] 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 *Proceedings of the National Information Systems Security Conference*. New York, NY: ACM Press, 1996.

[87] T. Toth and C. Kruegel, "Connection-history based anomaly detection," in *Proceedings of the IEEE Workshop on Information Assurance and Security*. Washington, DC: IEEE Computer Society, 2002, pp. 25–30.

[88] D. Ellis, J. Aiken, K. Attwood, and S. Tenaglia, "A behavioral approach to worm detection," in *Proceedings of the Workshop on Rapid Malcode*. New York, NY: ACM Press, 2004.

[89] T. Dubendorfer and B. Plattner, "Host behaviour based early detection of worm outbreaks in internet backbones," in *Proceedings of the IEEE International Workshop on Enabling Technologies: Infrastructures for Collaborative Enterprises (WETICE)*. Washington, DC: IEEE Computer Society, 2005, pp. 166–171.

[90] D. Brumley, J. Newsome, D. Song, H. Wang, and S. Jha, "Towards automatic generation of vulnerability-based signatures," in *Proceedings of the IEEE Symposium on Security and Privacy*. Washington, DC: IEEE Computer Society, 2006.

[91] N. Kawaguchi, H. Shigeno, and K.-i. Okada, "d-ACTM: Distributed anomaly connection tree method to detect silent worms," in *Proceedings of the IEEE International Conference on Malicious and Unwanted Software*. Washington, DC: IEEE Computer Society, April 2007, pp. 510–517.

[92] J. Mason, S. Small, F. Monrose, and G. MacManus, "English shellcode," in *Proceedings of the Conference on Computer and Communications Security*. New York, NY: ACM Press, 2009, pp. 524–533.

[93] J. Jung, R. Milito, and V. Paxson, "On the adaptive real-time detection of fast-propagating network worms," in *Proceedings of the Conference on Detection of Intrusions and Malware and Vulnerability Assessment*. Berlin, Heidelberg: Springer-Verlag, July 2007, pp. 175–192.

[94] M. P. Collins and M. K. Reiter, "Hit-list worm detection and bot identification in large networks using protocol graphs," in *Proceedings of the Symposium on Recent Advances in Intrusion Detection*. Berlin, Heidelberg: Springer-Verlag, September 2007, pp. 276–295.

[95] C. C. Zou, W. Gong, D. Towsley, and L. Gao, "The monitoring and early detection of Internet worms," *ACM Transactions on Networking*, 2005.

[96] S. Stafford, J. Li, T. Ehrenkranz, and P. Knickerbocker, "GLOWS: A high-fidelity worm simulator," University of Oregon, Tech. Rep. CIS-TR-2006-11, 2006.

[97] DETER, "DETER: Cyber defense technology experiment research (DETER) network," http://www.isi.edu/deter/.

[98] Lawrence Berkely National Laboratory, "LBNL/ICSI enterprise tracing project," http://www.icir.org/enterprise-tracing/, 2005.

[99] W. N. R. Group, "WAND WITS: Auckland-IV trace data," http://wand.cs.waikato.ac.nz/wand/wits/auck/4/, April 2001.

[100] University of Massachusetts Amherst, "Umass trace repository," http://traces.cs.umass.edu/, 2008. [Online]. Available: http://traces.cs.umass.edu/

[101] F. C. C. Osorio and Z. Klopman, "And you though you were safe after SLAMMER, not so, swarms not zombies present the greatest risk to our national Internet infrastructure," in *Proceedings of the IEEE International Conference on Malicious and Unwanted Software*. Washington, DC: IEEE Computer Society, April 2006, pp. 546–552.