WEWORM: AN ETHICAL WORM GENERATION TOOLKIT

Mark Wallis and Frans A. Henskens
School of Electrical Engineering and Computer Science, University of Newcastle, Newcastle, N.S.W, Australia

Keywords: Worm, Virus, Ethical, Toolkit, Cleanup, Infection.

Abstract: One of the largest threats to network security in Internet-connected corporate networks is attack via worms. Worms infiltrate the network and compromise security, consuming processor cycles and network bandwidth that would otherwise be available for corporate use. This research analyses a particular strain of network worm called the "Ethical worm", which is targeted towards beneficial means rather than malicious. Also presented is a design for an Ethical worm generation toolkit (named WEWorm) that will aid System administrators in the cleanup of an infected network.

1 INTRODUCTION

One of the largest threats to network security in Internet-connected corporate networks is attack via viruses and worms. While viruses are a continuing concern in the industry it is worms that are posing the greatest threat. Worms infiltrate the network and compromise security, consuming processing cycles and network bandwidth that would otherwise be available for corporate use. Additionally, they may disable or alter the operation of resources from their surrounding environment with the intent of hampering administrative efforts to detect and remove them. In a large corporate network that spans many distinct physical sites it is normally left to a single group of network administrators to detect and resolve worm or virus outbreaks across the corporation. One of the methods than can be used to deal with worm outbreaks involves the use of patch programs. When worms are flooding the communications links and stealing CPU cycles, administrators need a method by which such a patch program can be effectively and automatically distributed across the network - essentially mimicking the way that worm traffic can spread throughout the corporation. A patch program deployed in this way is called an Ethical worm.

This paper presents WEWorm, the outcome of a project that involved the design and development of prototype tools that allow the security team of an organisation to build and deploy Ethical worms. Such worms are able to move from machine to machine, curing any Malicious worm infections before propagating to other hosts. WEWorms have the ability to move between hosts using minimal resources as well as the ability to disguise themselves to avoid detection by sentient worms that are designed to be self-preserving.

2 PROBLEM DESCRIPTION

In 2001 the CodeRed worm was analysed and found to have infected more than 359,000 servers in less than 14 hours, with a peak infection rate of 2,000 new hosts every minute (CAIDA 2006).

In 2003 the Sapphire/Slammer worm had managed to spread itself worldwide in less than 10 minutes – making it the fastest spreading worm to date (CAIDA 2006).

In 2004 at the time of the W32/Mydoom.A worm infection it is estimated that one in 12 emails on the Internet were infected with the MyDoom virus (SecurityStats undated).

The statistics above give an insight into one of the largest security concerns that exists on the Internet today. Malicious programmers are developing worms and viruses that can spread and contaminate at alarming rates. Security administrators do not have the time and resources to be able to act upon the results of existing detection and notification systems before their network becomes compromised with each latest viral outbreak.

Much research is currently active in the area of detection and automatic-response for computer virus and worm infections (C.C Zou 2003; S. Sidiroglou 2003). Unfortunately the skills of the worm developers are now at a level where the efforts of the network administrators are focused more on cleaning up network infections, than preventing them. No longer can it be assumed that strong network perimeter defences are enough to protect a network from infection. Another important issue involves the problems and restrictions placed on a network when the administrator is tasked with cleaning up an infected environment. For example, the performance of an infected network may well work against attempts to remove the infection. The situation becomes even more difficult when the infection involves malicious software that is actively trying to ensure its own survival. Such Malicious software may be capable of disabling local host virus protection and other secondary systems that remote administrators use to aid the cleanup process.

A growing number of corporate private networks contain multiple geographically disparate sites connected by wide area links. In such situations a single group of network administrators may well be tasked with the cleanup of nodes over the diverse environment. Unfortunately many of the latest generation of Malicious worms generate so much network traffic in their attempt to spread that they flood the communications links that administrators use to remotely manage distant nodes. The wide area communication links to distant nodes are of necessity slower than local links and reduction in their performance is particularly damaging.

The focus WEWorm is the cleanup of environment-wide infections in private networks – such as a network owned by a single corporation, or a single educational institution.

2.1 Current Opinions

2.1.1 V. Bontchev - “Are ‘Good’ Computer Viruses still a Bad Idea”

In (Bontchev 1999) the author details the particular reasons why the greater security community sees Ethical worms as harmful. Not all points are relevant to all Ethical worm designs, but it is beneficial to review them to provide a requirement from which to build a design.

The reasons are summarised as follows:

1. Lack of control – once an Ethical worm is released into the wild there is no control mechanism by which an administrator can remotely choose to halt its execution.

2. Recognition difficulty – if an Ethical worm spreads via the same distribution mechanism as a Malicious worm, existing anti-virus software may not be able to tell them apart and act negatively towards the Ethical worm.

3. Resource wasting – an Ethical worm may use up computer and network resources just as a Malicious worm can.

4. Compatibility problems – it is impossible to test every combination of hardware and software in hosts connected to the Internet – hence an Ethical worm may hit compatibility problems trying to execute on any non-standard hosts it infects.

5. Effectiveness – under certain circumstances it can be found that a non self-replicating piece of software would perform the same task as an Ethical worm.

6. Unauthorised access – just like their malicious counterparts, Ethical worms tend to execute on the host system without authorisation from the host owner.

7. Misuse – any sort of system designed for deploying Ethical worms could also be used to then also design and deliver Malicious worms

8. Trust – any worm, no matter its intention, is capable of removing the user’s trust in their computer system, because the user can no longer guarantee that they are familiar with every piece of software executing on their system.

9. Negative Common Meaning – due to the media spin of the words “virus” and “worm” it is very hard for normal users to understand the idea of an Ethical worm whose aim is to do good.

2.1.2 B. Schneier – “Benevolent Worms”

In (Schneier 2003) Schneier claims that Ethical worms are essentially patch-delivery mechanisms, and as such should be designed to cover the following four major characteristics of a good patch-delivery system:

1. People can choose the patches they want,

2. Installation is adapted to the host it is executing on,

3. It is easy to stop an installation in progress, or uninstall the software,

4. It is easy to know what has been installed where.

Historically, Ethical worms have addressed none of the above items – and hence their effectiveness in solving the worm problem has come under scrutiny.

3 WEWORM PROJECT

As can be seen from the above reviews, the Ethical worm concept is not strongly supported. Initially it was intended that WEWorm would be used to clean up worms in Internets spanning multiple private networks. In the light of current commentary, it was decided, instead, to focus on a particular restriction to the scope of this original intention. This allows the design to address the majority of the issues identified by previous research, and to develop a scalable solution to the problem of the repair and clean up of infected hosts.

To date all the reported Ethical worms have been developed in a way that ensured they spread across as many hosts and networks as possible – just like the Malicious worm they were trying to remove. The WEWorm design presented in this paper aims to target small networks of hosts controlled as a single group. This provides a solution for use by large corporations and organizations for which a priority is the removal of worms from their own internal networks, rather than the removal of a worm from all hosts on the Internet.

This restriction introduces new issues that need to be addressed – namely how an administrator can control a worm to guarantee it will not spread outside of the network into which it was introduced to protect.

3.1 Effect of Domain Restriction

This section reviews the criticisms raised in (Bontchev 1999) and discusses the restriction of the problem domain to private networks.

<u>Lack of control</u>: Due to the restricted scope of the problem domain described above, any control mechanism will be much more effective. This is because the Ethical worm no longer has to communicate with active instances over the global Internet. A control mechanism based on a modular framework was developed. This framework provides an effective mechanism for passing control messages between the active Ethical worm instances, and provides a level of active control over the system.

<u>Recognition difficulty</u>: Limiting the scope of Ethical worms to a single network makes it possible to create a unique propagation method for it. This allows the worm to spread through the network using a sanctioned propagation technique that is unique to Ethical worms in this environment. Combined with various encryption signatures described later in the paper, this provides an excellent method of distinguishing sanctioned Ethical worms from Malicious worm infections.

<u>Resource Wastage</u>: Problem domain restriction negates the issue of resource wasting by Ethical worms. If an Ethical worm is sanctioned and deployed by a network administrator in a single network entity then any resources used by that Ethical worm will be well known and accepted before deployment. It will be shown that the WEWorm toolkit can be used to provide estimates of resource usage to the Ethical worm designer to help him/her understand the resources required to deploy the self-propagating code on their network.

<u>Compatibility Problems</u>: Deploying an Ethical worm into a single well-defined network entity gives the worm designer the ability to tailor the worm to specific subsets of infected nodes. Compatibility problems can be mitigated in networks where a standard SOE (Standard Operating Environment) deployment is used on all machines. The toolkit design will show how an Ethical worm can be designed and then test-deployed on a standard SOE with a view to allowing network administrators to complete a full test cycle before the work is released into the network.

<u>Effectiveness</u>: While the problem domain restriction does not specifically address the issue of effectiveness, a comparison will be made between the restricted technique and other mainstream patch-delivery systems that can be used to solve similar problems.

<u>Unauthorised access</u>: Using WEWorm, the only person who will be able to deploy an Ethical worm will be the system administrator responsible for the network. Focusing on single-entity networks negates the issue of ownership of the systems. If the single entity network actually contains public clients (i.e. a university network that contains student-owned machines) then the users may be made aware of the possible deployment of Ethical worms in the terms and conditions of use of the network.

<u>Misuse</u>: WEWorm addresses the issue of misuse by ensuring that any worm developed using the toolkit is designed in such a fashion that can only propagate to nodes that trust the signature of the worm's author. This ensures that the design cannot be directly reused for malicious intent without further alteration to remove safeguards.

Trust: As with the issue of unauthorised access, only Ethical worms designed by the network administrator can propagate to nodes in the network deploying the presented design. This ensures that the end user's trust in their workstation is maintained and is not compromised any more than by other automatic patch delivery mechanisms.

Negative Common Meaning: By restricting the problem to single-entity networks it can be ensured that appropriate training and documentation can be developed and circulated among users and staff. The purpose is so they are aware that the Ethical worm deployment is a trusted design by their administrative staff, and should be trusted despite the negative connotations of the name.

3.2 Design Goals

The following major design goals were identified for the WEWorm project. These goals also include a subset from (Schneier 2003), in which the author describes a minimum set of requirements that any patch deployment system has to exhibit to be trusted and effective.

Identification of infection: The system must be designed in such a way that it can be further integrated directly into a larger Intrusion Detection system. This would allow IDS alerts to automatically trigger Ethical worm actions and deployment.

Responding to infection: External intrusion detection systems must be able to communicate with the Ethical worm system to allow automatic deployment of Ethical worms from an existing library. The IDS would be able to store a link between a certain signature and its related Ethical worm to automatically combat infection.

System Security: The Ethical worm system must be secure from abuse by Malicious entities. This security must protect the system from being reused for the distribution and development of Malicious worms while also providing security of the overall environment.

Patch choice: Users should be able to choose which patches are deployed on their system. The restricted implementation domain allows the network administrator to decide which Ethical worms will be deployed into the network. While the end-user will continue to have no choice, in such environments security is not their responsibility and hence the trust and decision is placed on the network administrator to choose the appropriate patches to be deployed to each system.

Adaptive Installation: Each patch should be adaptive to the host on which it is being deployed. It will be shown that the presented modular WEWorm design allows the administrator to develop Ethical worms that act differently depending on the host on which they are executing. This will give added power, allowing the Ethical worm to work across multiple installations and architectures as required.

Installation Interruption: A patch system must be interruptible, even when it is halfway through a patch installation. It will be shown that the control methods in the WEWorm design allow the user to control and stop the further propagation of an Ethical worm in the network. Simulation and testing demonstrate that the system does this effectively.

Installation Reporting: The WEWorm system implements a monitoring and reporting mechanism that satisfies the requirement that any patch management system must be able to report on what patches have been installed, and where in the network. Similarly to some of the historical Ethical worms described previously, the design allows the user to log the worm's actions both on the host itself and to a centralised database. These logs record when a particular host executes an Ethical worm, after the network administrators have deployed it.

3.3 Design of WeWorm

WEWorm implements an Ethical worm system targeted at the cleanup of Malicious worm infections in a single network entity. The above design considerations were used to define the system requirements, and the above criteria were used to create a design that addresses the documented concerns about the Ethical worm concept, while also meeting all the specified design goals. The overall design was based on a toolkit concept that allows a network administrator to rapidly design, build and deploy an Ethical worm into a network environment. The developed Ethical worm is supported by a pre-established runtime environment.

3.3.1 Toolkit Concept

The modular design of an Ethical worm is well suited to the use of a component based architecture that allows rapid, piece-by-piece building of a solution. The toolkit is constructed using two groups of components – common and targeted. Some components form part of the worm itself and some execute on the target hosts as part of the environment that supports the Ethical worms. This component architecture also allows corporations to share components between each other to remove a worm infection on a more global scale.

Components are classified as follows based on their role in the Ethical worm system.

3.3.2 Common Components Groups

Common components are those that are shared between, and required for, the operation of all Ethical worms. These components comprise the core of the payload and interact with the runtime environment to provide end-to-end connectivity for the system. Common components are of two kinds. The first kind is packaged as part of each WEWorm instance and is used when the worm is being executed in the runtime environment. The second kind is packaged as part of the environment itself as is used in handling the processing of each new WEWorm instance, as it is executed within the environment.

The common components that are packaged with WEWorm instances are the:

- Propagation Component
- Outgoing Security Component
- Outgoing Network Component
- Reporting Component
- Control Component

The common components that are packaged with the runtime environment include:

- Incoming Security Component
- Incoming Network Component

WEWorm instances also include targeted components that are tailored to deal with the Malicious worm for which they are designed.

3.3.3 Targeted Components

Targeted components deal with specific Malicious worm attacks and generally need to be modified or created specifically for each Ethical worm that is deployed. The main targeted component is the Executable component.

3.3.4 System Walkthrough

The following section describes the process followed to create and deploy an Ethical worm. It demonstrates the steps that a Network Administrator takes to use the WEWorm system to cleanse an infection in a local network.

1. Network administrator receives an alert from their IDS system that an infection has occurred.
2. Network administrator uses this IDS information to ascertain the nature of the infection.
3. Network administrator designs the requirements for the Ethical worm based on the IDS information.
4. Network administrator builds the WEWorm payload using the pluggable modules. If required a targeted executable component is developed to address a specific instance of Malicious worm.
5. Network administrator manually loads the payload into a host in the network and executes it.
6. Normal execution flow for the payload begins:
   - 6.1 The Incoming Network component in the node's environment receives the payload.
   - 6.2 The Incoming Security component in the node's environment authenticates the payload.
   - 6.3 Control Component executes to ensure there are no administrative commands waiting to be actioned.
   - 6.4 Executable component of the payload executes a single command.
   - 6.5 If more commands exist then repeat through Control and Executable tasks until all commands have been executed.
   - 6.6 Reporting component executes.
   - 6.7 Propagation component executes and if further propagation is required then:
     - I. payload for propagation is created
     - II. outgoing Network component delivers the payload to the next node.

3.4 Component Development

The targeted component development follows the previously described toolkit methodology. The task of cleaning up a Malicious worm infection can be seen as a series of common steps with varying parameters. Implementing a subset of these commands allows for rapid development of the actual Ethical payload as well as the surrounding support infrastructure.

The following are examples of modular commands that can be implemented using WEWorm. They are a subset of the commands used by real-world viral removal instructions.

- Kill process
- Remove registry parameter
- Delete file
- Reboot host
- Download file via TFTP
- Execute file

For example, to remove a Blaster infection the following payload commands would be sequenced together

- Download the Microsoft patch from a TFTP server
- Execute the Microsoft patch
- Kill msblast.exe process
- Delete msblast.exe file from System32 folder
- Delete registry key HKLM\Software\Microsoft\Windows\CurrentVersion\Run, value windows auto update

3.5 Delivery Mechanism

Design of WEWorm included analysis into the best network delivery mechanism that could be used for propagating Ethical worm payloads between nodes in a network environment. The delivery mechanism has to satisfy specific requirements to be acceptable for use in hostile and congested networks. These requirements include

- High levels of efficiency,
- Small packet sizes, and
- Focus on local domains.

High levels of efficiency and small packet sizes are requirements that specifically address the problem if communication in a network that is congested with Malicious worm traffic.

Taking these requirements into account the following network protocol was specified to handle inter-node communication:

```
IP Protocol: UDP
Source Port: dynamic
Destination Port: 55
Addressing Method: 1:m Broadcasting
Packet Payload: dynamic packet length
based on tuning parameters suited towards
a specific network outbreak.
```

UDP was chosen as the transport protocol due to its lightweight implementation and ability to pass data without the added network bandwidth requirements of hand shaking. The initiating node dynamically chooses the source port. The destination port is deliberately chosen to be less than 1024, thus ensuring that only privileged processes can bind to that port. The addressing method chosen is a one-to-many broadcast, which as described earlier allows a single host to propagate to as many other hosts in the same broadcast domain as possible. In a perfect network, every host in the domain should receive a one-to-many broadcast payload. However, in the presence of malicious infections, it is possible that this is not always the case, and hence multiple stages of propagation are supported in a fashion that models the spread of Malicious worms themselves. The broadcast delivery mechanism also ensures that the design is focused on local domains as per the project scope definition (noting that IP only supports intra-network broadcast).

The use of a broadcast delivery mechanism introduces the constraint that an Ethical worm can never propagate to a node outside of the current broadcast domain. This acts as a safe guard, but also introduces issues with large corporate networks that span multiple broadcast domains. These issues are solved using repeater nodes that essentially tunnel the broadcast traffic between domains.

The use of a custom UDP packet design also allows the user of WEWorm technology to choose how to structure packets so they are able to traverse a malicious network. A trade off may be required between the size of packets and the number of packets it can take to deliver a payload. The more packets required to propagate a payload the more CPU power is required on the end nodes to authenticate and rebuild the final payload deliverable. Increases in the number of packets also means that each packet is smaller and more likely to be able to move through congested networks than large fragmented packets. This trade off is a configurable option of the outgoing delivery component.

3.6 Delivery Security

Security is paramount in the environment, thus ensuring that only trusted sources are able to disseminate Ethical worms. Special caution needs to be taken to ensure the deployment environment cannot be used for malicious means. The WEWorm design relies on signing and encryption methods that are optional components of deployed Ethical worms. A trade-off is available to be made by the network administrator for security versus efficiency.

A high security option is available that encrypts each outgoing packet prior to processing by the delivery component. This encryption ensures that not only is the payload data protected from sniffing attacks while in transit over the network, but also that payloads encrypted with the trusted key are the only ones that can be executed on each node. The encryption is based on public/private key encryption with each node in the system having a list of acceptable public keys from which to accept payloads.

The medium security option implements a security component that does not encrypt, but signs each packet prior to processing. This allows the receiving node to validate that the payload has come from a trusted source but will not protect the payload itself from being viewed by malicious entities on the network.

The low security option sends packets in raw form, without any encryption or validation mechanism.

4 WEWORM PROTOTYPE

A prototype of WEWorm has been implemented. This prototype provides the key functionality required to develop and deploy an Ethical worm that is targeted at cleaning up a Malicious worm infection. The following design choices were made for this prototype.

Choice of Runtime: A user-mode software runtime solution was chosen due to the high levels of rapid prototyping afforded by such an environment. While a user-mode software solution is not the most secure option due to its susceptibility to local malicious code attacks, it does provide the greatest flexibility when it comes to interacting with the underlying operating system of the node in order to remove an infection.

Choice of Language: Java was chosen as the development language for the prototype. This is largely due to the fact that the Java Virtual Machine (JVM) (T. Lindhalm) essentially mimics the user-mode software runtime environment specified in the initial design. Each node in the network executes a JVM instance which provides the environment in which the Ethical payload will execute.

Choice of Environment: The prototype was developed in a small-scale real network environment consisting of three servers executing multiple virtual machines. This created a virtual network of 15 infected hosts configurable in various network deployments – including a large single broadcast domain and two separate broadcast domains separated by a router and VPN.

Runtime Environment Overview: The runtime environment was implemented as an application running under a JVM on each node in the network. This application executes code that listens for all incoming packets on the specified UDP port. Once a data packet is received the runtime environment uses streamlined pipeline architecture to process the Ethical worm payload. Each component in the process is deployed as part of a library of features in the runtime environment. If any of the components required to execute a payload are not available then the complete payload is rejected for execution.

Manual control of the runtime environment is available using a subset of command line tools that allow the administrator to perform tasks such as manually executing a payload and reporting the current environment's status.

The JVM design was chosen to give a high level of support for deploying WEWorm Ethical worms into a heterogenous environment. Creation of a specific Virtual Machine for WEWorm deployment was outside the scope of a prototype, and additionally was unnecessary due to the superset of the required functionality that the JVM is able to provide

4.1 Testing and Simulation

Testing of the prototype was broken into two main categories – live testing and simulation.
Live testing was completed using real hardware executing multiple virtual hosts to mimic a small corporate network. A Cisco router was introduced to allow testing of the prototype across multiple broadcast domains. The tests involved infecting all hosts in the network with a "captured" Malicious worm and then deploying an Ethical worm, built with the toolkit, to remove the infection. The Ethical worms used for testing were built with reporting components that allowed a central host to track the Ethical worm in various network scenarios. Scenarios were chosen to simulate differing levels of network usage and domain structure.

The following live tests were completed:

1. Blaster infection across 15 hosts in a single broadcast domain. Network usage levels normal.
2. Blaster infection across 15 hosts in a single broadcast domain. Network flooding enabled to create high packet loss.
3. Blaster infection across 15 hosts in two broadcast domains with a pair of repeater nodes used to replicate the deployment. Network usage levels normal.
4. Blaster infection across 15 hosts in two broadcast domains with a pair of repeater nodes used to replicate the deployment.

Network flooding, enabled between the broadcast domains, was used to create high packet loss.

After the live test results were gathered the data was used to extrapolate results for larger networks using a network simulation package. The ns-2 network simulator (ISI) package was used to create large but basic scenarios in order to test propagation between large numbers of nodes in single broadcast domains.

The anti-Blaster Ethical worm developed for this testing was created from the following components

- Propagation Component = broadcast run-once with delay.
- Security Component = message signing, no encryption.
- Network Component = 1:m broadcasting.
- Reporting Component = centralised HTTP POST report.
- Control Component = centralised HTTP GET control.

4.2 Test Results and Review

The following results were gathered on the test scenarios described above. All tests were executed using the standard packet strategy of large packet size with no fragmentation. The resulting packet size of the payload was 5.1 kb's.

Table 1: Test results with large packet size.

Test Type	**Nodes**	**Domains**	**Packet Loss**	**Total time**	**Nodes per sec**
Real	15	1	None	6 secs	2.5
Real	15	1	75%	34 secs	0.44
Real	15	2 (routed)	None	8 secs	1.88
Real	15	2 (routed)	75%	62 secs	0.24
Simulated	50	1	None	~ 25 secs	2
Simulated	100	1	None	~ 50 secs	2
Simulated	1000	1	None	~ 500 secs	2

The real prototype environment was rebalanced to use smaller packets for payload delivery. The packets were reduced to 1 kb so it took 6 packets in total to deliver the WEWorm. The following results were obtained:

Table 2: Test results with small packet size.

Test Type	**Nodes**	**Domains**	**Packet Loss**	**Overall Time**	**Nodes per Second**
Real	15	1	None	14 secs	1.07
Real	15	1	75%	31 secs	0.48
Real	15	2 (routed)	None	19 secs	0.79
Real	15	2 (routed)	75%	54 secs	0.28

5 CONCLUSION

The community does not currently embrace the concept of the Ethical worm. This research has shown that, with appropriate domain restriction and design, the concept has a place in modern network security.

Testing results, both real and simulated, have shown that Ethical worms provide the ability to quickly clean up malicious infections, and that they provide environmental scalability.

The design has met all of the major original requirements set out in (Bontchev 1999). In review

- Lack of control – resolved using control modules and direct interaction with the runtime environment on the host
- Recognition difficulty – resolved by looking for an authorised signature that verifies the worm payload as authentic
- Resource wasting – resolved using a very light-weight UDP based protocol and streamlined execution pipeline
- Compatibility problems – resolved by domain restriction
- Effectiveness – addressed by inherent worm design that can handle largely unstable networks
- Unauthorised access – resolved by domain restriction
- Misuse – resolved using payload signing and encryption to ensure nodes only execute authentic payload's
- Trust – resolved by domain restriction

The initial design goals have also been met as follows:

- Identification of infection – resolved by the creation of an integration point for IDS systems to design WEWorms.

- Responding to infection – resolved by the creation of an integration point for IDS systems to generate WEWorms.
- System security – resolved by the Security modules that handle the encryption and network delivery of WEWorms.
- Patch choice – resolved by allowing the Network Administrator to use a 1:1 delivery mechanism to control WEWorm propagation
- Adaptive Installation – resolved by allowing the Network Administrator to use a 1:1 delivery mechanism to target WEWorms towards subsets of hosts.
- Installation Interruption – resolved by continuous Control module polling during the WEWorm execution cycle
- Installation Reporting – resolved by the Reporting module of the WEWorm design.

There is scope for much more work in this field. The design phase of WEWorm has shown that a user-mode software solution, while being well suited for prototyping, is not the most suitable for actual deployment. Work in pushing the Ethical worm support into kernel space, or even into a hardware implementation would greatly improve the systems resistance to attack.

5.1 Future Work

This research suggests that the following items warrant further investigation and analysis.

Active Deployment: Active deployment is a topic that involves IDS systems automatically generating and releasing Ethical worms to combat infection in real time with no interaction required for the System Administrator. Further development of WEWorm to integrate deployment of Ethical worms into existing IDS products would result in a powerful end-to-end solution.

Provision of a GUI: At this stage the Ethical worm payload is described using an XML schema. Writing of XML is notoriously error-prone, so automation of generation is desirable. WEWorm has been designed and implemented to support ultimate provision of a Graphical User Interface. Such a GUI would allow network administrators to easily drag-and-drop components to build up this XML representation, ready for compilation using the existing utilities.

Runtime Development: The current WEWorm prototype was developed using the user-mode software runtime concept. It is known that this approach has security issues related to protection of the runtime environment itself from malicious attack. Further development leading to hardware that implements this runtime environment would increase security and reduce load on the node processors. A preliminary design of such hardware was been completed, as an additional module for integration into standard network interface cards (NICs).

REFERENCES

Bontchev, V. (1999). "Are ‘Good’ Computer Viruses Still a Bad Idea ?" from http://vx.netlux.org/lib/avb02.html.

C.C Zou, L. G., et al. (2003). "Monitoring and Early Warning for Internet Worms." from http://www-unix.ecs.umass.edu/~gong/papers/monitoringEarlyWarning.pdf.

CAIDA. (2006). "Code-Red Security Analysis." from http://www.caida.org/analysis/security/code-red/.

CAIDA. (2006). "Sapphire Security Analysis." from http://www.caida.org/analysis/security/sapphire/.

ISI. "The Network Simulator - ns-2." from http://www.isi.edu/nsnam/ns/.

S. Sidiroglou, A. K. (2003). "Countering Network Worms through Automatic Patch Generation." from http://www.cs.columbia.edu/techreports/cucs-029-03.pdf.

Schneier, B. (2003). "Benevolent Worms " Crypto-Gram Newsletter Retrieved Sep 03, from http://www.schneier.com/crypto-gram-0309.html#8.

SecurityStats. (undated). "Virus Statistics." from http://www.securitystats.com/virusstats.htm.

T. Lindhalm, F. Y. "The Java Virtual Machine Specification." from http://java.sun.com/docs/books/vmspec/2nd-edition/html/VMSpecTOC.doc.html.

APPENDIX: XML DESCRIPTOR

An example of the XML descriptor used to build a worm using this toolkit can be found at the link below:

http://mark.serialmonkey.com/objects/sample_xml.txt