

INFORMATION SOCIETY TECHNOLOGIES (IST)
PROGRAMME



Large Scale Monitoring of BroadBand Internet Infrastructure
Contract No. 004336

D3.3b: Overview of potential applications using the LOBSTER system

Abstract:

This document describes the new potential LOBSTER application or new potential extensions to existing LOBSTER applications.

| | |
|------------------------------|------------------|
| Contractual Date of Delivery | 31 December 2006 |
| Actual Date of Delivery | 20 December 2006 |
| Deliverable Security Class | Public |

The LOBSTER Consortium consists of:

| | | |
|----------|----------------------|-----------------|
| FORTH | Coordinator | Greece |
| VU | Principal Contractor | The Netherlands |
| CESNET | Principal Contractor | Czech Republic |
| UNINETT | Principal Contractor | Norway |
| SYMANTEC | Principal Contractor | United Kingdom |
| ALCATEL | Principal Contractor | France |
| FORTHnet | Principal Contractor | Greece |
| TNO | Principal Contractor | The Netherlands |
| TERENA | Principal Contractor | The Netherlands |



Contents

| | | |
|----------|--|-----------|
| 1 | Introduction | 5 |
| 2 | Existing LOBSTER application | 7 |
| 2.1 | Appmon | 7 |
| 2.2 | ABW | 7 |
| 2.3 | EAR Live | 8 |
| 2.4 | Extended NetFlow | 8 |
| 2.5 | LOBSTER Network Telescope | 9 |
| 2.6 | PacketLoss | 9 |
| 2.7 | Sub-Second Bandwidth | 9 |
| 2.8 | Stager | 9 |
| 3 | New potential applications | 11 |
| 3.1 | Extending EAR Live | 11 |
| 3.2 | Extending Extended NetFlow | 11 |
| 3.3 | Security toolbox | 12 |
| 3.4 | Visualization of geographical characteristics of network traffic | 12 |
| 3.5 | Monitoring Wikipedia traffic | 13 |
| 3.6 | Per-application Network Traffic Characterization and Throttling | 14 |
| 3.6.1 | Overview | 14 |
| 3.6.2 | Extending Appmon for Per-application Traffic Throttling | 14 |
| 3.7 | Polymorphic Attack Detection | 15 |
| 3.7.1 | Introduction | 15 |
| 3.7.2 | Polymorphic Shellcode Detection using Emulation | 16 |
| 3.7.3 | Integration with LOBSTER | 17 |
| 4 | Summary | 19 |

CONTENTS

Chapter 1

Introduction

In this deliverable we give a short description of new potential applications and utilities that take advantage of the LOBSTER passive monitoring framework. These applications are either completely new or they are potential extensions to the applications already implemented by the LOBSTER consortium.

In chapter 2 a quick overview of existing LOBSTER applications are given and in chapter 3 the new potential applications or extensions to existing applications are described. The new applications that are being described are:

- Extensions to the Extended NetFlow application.
- Extensions to EAR Live
- Security toolbox
- Visualization of geographical characteristics of network traffic
- Monitoring Wikipedia traffic
- Per-application Network Traffic Characterization and Throttling
- Polymorphic Attack Detection

Chapter 2

Existing LOBSTER application

This chapter contains a short overview of the existing LOBSTER applications that have already been implemented by the LOBSTER partners. A more detailed description is available in “D3.1 LOBSTER Applications”.

2.1 Appmon

Appmon is an application for network traffic characterization. It splits the consumed bandwidth to the applications that generated it and identifies the top hosts with the highest bandwidth consumption.

In order to achieve identification of network traffic we use three different approaches for different kind of applications. First, for simple applications we use BPF filters in order to categorize them. For more complex applications we use more sophisticated trackers that deploy deep packet inspection in two forms. The one form is to look inside the application message for characteristic protocol patterns. The other form is to fully decode the application protocol to identify the new, dynamically generated port, which is going to be used and categorize the corresponding network flow to belong to the application.

2.2 ABW

ABW application monitors how bandwidth usage on a network link is divided into various protocols in different layers corresponding to OSI model (e.g., transport layer or application layer).

ABW is written on top of DiMAPI (Distributed Monitoring Application Interface) and the trackflib library to track application protocols. The ABW application itself was developed as part of the JRA1 activity of the GN2 project. DiMAPI and the trackflib library were developed by LOBSTER.

Some highlights for the ABW application are:

- Computes bandwidth used by different protocols in different layers, also shows percentage of IPv6 and multicast traffic
- Indicates bandwidth usage in different timescales including short-term samples (1 second averages) in order to detect short peaks
- Shows real bandwidth dynamics on a monitored line (not after stressing user traffic with performance test)

- Measurement can be frequent and continuous
- Can compute bandwidth of user-defined flows (specified by header filtering and payload searching)

2.3 EAR Live

EAR Live is an application for zero-day Internet worm detection based on the identification of packets with similar contents directed to multiple destination hosts. EAR's detection method is based on four observations which are commonly found in known worms:

- Diversity of Destinations: The network packets that belong to the same worm tend to have a very large number of destinations. Actually, this seems to be an inherent property of all the worms: worms tend to spread to as many victims as possible, and therefore, their network packets seem to have a large number of destinations.
- Spread by Clients: Most worms are usually spread by clients, i.e. by computers that initiate a (usually TCP) connection. This property, as well, seems to be an inherent property of aggressive worms. Indeed, in order for a worm to spread fast, it needs to initiate connections to its potential victims, rather than to wait for the potential victims to connect to it.
- Payload Repetition: Several of the network packets that belong to the same worm, tend to contain similar (if not identical) payloads.
- Small Size: Worms tend to be small in size, in order to spread as fast as possible. A large worm size would prolong infection time and consume bandwidth that could be used for infecting other targets. Well-known worms such as CODE-RED, Blaster and Welchia, Sapphire, and the Witty worm have all the above four properties.

2.4 Extended NetFlow

The Extended NetFlow application uses the IPFIX protocol to export flow records that contains several new additional flow record attributes not found in standard IPFIX. Most of these extra attributes are related to QoS measurements:

pktLenHistogram a histogram of the length of packets in a flow

pktDistHistogram a histogram of the distance between packets in a flow

pktLength[Var/Sum/SumQ] statistical values for length of packets in the flow

pktDist[Var/Sum/SumQ] statistical values for the distance between packets in the flow

direction provides information about who initiated the TCP connection in the flow

reordered number of bytes that are out of sequence in a TCP flow

maxRate[1s/100ms/10ms/1ms] information about the maximum bit rate in the intervals

minRate[1s/100ms/10ms/1ms] information about the minimum bit rate in the intervals

service looks into the payload of packets to give information about the application that generates the traffic. Recognizes P2P applications like Bittorrent, eDonkey and Gnutella.

A flow collector and reporting tool that supports the extra attributes have also been developed. The reporting tool can generate high level report like which AS number has the highest number of packets out of order or which long lasting flow has the highest jitter? These reports are then inserted into Stager for long term statistics.

2.5 LOBSTER Network Telescope

The LONT application is a tracking service for suspicious Internet traffic. A user of the service specifies a description of suspect traffic and the desired form of post processing and results. The LONT service translates this description into MAPI calls and instructs LOBSTER nodes to measure the network for the specified traffic. Results of the measurements (possibly after conversion) are shown to the user.

2.6 PacketLoss

An application for passive end-to-end packet loss measurement, which relies on traffic monitoring by two monitoring sensors at the two ends. PacketLoss uses a pair of remote passive monitoring sensors and runs on a third host. It is based on a new MAPI function, named EXPIRED_FLOWS, which counts the packets captured per flow and returns this information after the flow is expired. We consider a flow expired when it is not active for 30 seconds. The PacketLoss application receives expired flows from remote sensors and for the flows that match, it calculates the packets lost.

2.7 Sub-Second Bandwidth

The Sub-Second Bandwidth (SSB) application is a newly developed application not described in earlier deliverables. This is a relatively simple application that measures the sub-second bandwidth and packet rate in five minute intervals. Every five minute the application returns the total number of bytes and packets as well as the maximum, minimum and standard deviation for number of bytes and packets in the intervals 1 second, 100 millisecond and 10 millisecond. These values are then stored in the Stager application for presentation.

2.8 Stager

Stager is a system for aggregating and presenting network statistics. Stager is generic and can be customized to present and process any kind of network statistics. The backend collects data and stores reports in a database, automatically handling the aggregation of hourly statistics into days, weeks, and months. The Web frontend presents data in tables, matrices, or plots and the displayed reports are fully customizable.

So far Stager has been integrated with the LOBSTER applications Appmon, SSB and Extended NetFlow.

Chapter 3

New potential applications

This chapter contains a short description of some new potential applications or extensions to existing applications. This is by no means a complete list of applications that can run on the LOBSTER framework, but should be considered as a small example of what the framework can be used for in the future.

3.1 Extending EAR Live

EAR Live provides a powerful capability to very efficiently detect recurring strings longer than 200 characters. With minor modification, and the extensions described below, it might be possible to use the EAR Live tool to detect several forms of malicious code and malicious collusion, even if the longest common subsequence is less than 200 characters.

The primary challenge in doing this stems from the reality that many short strings naturally recur frequently in network traffic. However, machine learning tools could be used to capture the set of shorter strings that are normal for a given site, enabling focus on new and emerging recurrent short strings. Alternatively, frequency of occurrence could be recorded for shorter strings, and shorter strings with statistically significant sudden increases in frequency of occurrence could be highlighted for similar attention.

In both cases, EAR Live powerful efficiency in identifying recurrent strings forms a foundation extended with additional techniques. Such a hybrid system might prove powerful and effective in detecting botnet command and control traffic, spyware, and malicious code of various forms, even if the longest common subsequence is substantially shorter than 200 characters.

3.2 Extending Extended NetFlow

So far the Extended NetFlow application has added relatively generic quality of service related attributes to the IPFIX flow records. While these records makes it possible to create several new and novel reports not previously available from standard IPFIX, there are still room for improvements by adding new application specific attributes. This could for example be attributes that says something about the perceived quality users experience with SIP conversations, MPEG streaming of video or video conferences.

The generic jitter statistics already implemented in the Extended NetFlow application can give some information, but for both SIP and MPEG some jitter and packet loss can be acceptable. So the

Extended NetFlow application could be extended so that it detects SIP and MPEG and then analyze the traffic to see if the quality is considered acceptable or not.

It would also be possible to do more detailed per flow TCP statistics summing up properties like retransmissions, clustering and window sizes. Small receive windows will limit performance on high bandwidth*delay paths. Clustering comes from bursty traffic possibly caused by queuing delays.

Usage of the extended flow measurements for quality could be used to create new payment models where you pay for the throughput you use, generally or for streaming services for example. Your bill could be broken up by quality delivered and you would pay less for poorer actual quality.

3.3 Security toolbox

Most of the applications developed so far for the LOBSTER framework are relatively complex systems that need special installation. Most of them also demands so many resources that only one application can be active at one time.

The LOBSTER framework can be very useful for a lot of security related tasks and a set of lightweight tools could be developed to help security people investigate security incidents. Some usage areas for these tools are:

- Evidence collection - during an attack or suspicion about hacking attempts a LOBSTER probe can be used for evidence collection by capturing whole packets including payload.
- IPv6 - the use of IPv6 is slowly increasing and with the release of Windows Vista where IPv6 is turned on by default, it will most likely continue to increase. The problem is that many routers today have several limitations in monitoring IPv6 traffic like generating NetFlow containing the IPv6 traffic. The LOBSTER probes are fully IPv6 compliant and can be used for monitoring IPv6 traffic.
- Tunneled traffic - using LOBSTER probes it is also possible to look into and monitor tunneled traffic. A utility that detects tunneled traffic would have to be implemented.

A more resource heavy tool could also be traceback of traffic with spoofed source IP addresses. With a large LOBSTER infrastructure it would be possible to ask the various probes if they have seen a specific packet, and that way it would be possible to trace the origin of a packet.

3.4 Visualization of geographical characteristics of network traffic

For network planning it is useful to know relations of certain network traffic characteristics to geographical locations. For example, the volume of transferred data to and from different countries.

We propose an application that will combine two inputs:

- Network characteristics obtained from passive monitoring
- Geographical database of IP addresses

and that will then use a GIS (Geographical Information System) to visualize the relations of network characteristics to geographical locations for selected time periods.

More technical details of the application follow:

- A suitable GIS will be selected, many currently available systems of this category are listed at <http://www.maptools.org> or <http://www.freegis.org>.
- The application should allow alternative visualizations, such as the following:
 - different colors of states on the map
 - lines of different colors, thickness or styles between source and destination states
 - columns of different colors or thickness placed on top of states on the map (multiple columns can be placed on each state to visualize more characteristics)
- A regular web browser should be sufficient for the user to use the application
- Geographical aggregation should be possible, for example, to show characteristics per continent, rather than per country
- Proposed characteristics to visualize are the following:
 - the number of bytes of all connections
 - the number of bytes of selected applications or the number of connections
 - order of most frequently used applications
 - average (or median) and maximum value per connection of:
 - * duration
 - * the number of bytes
 - * throughput
 - * max. rwin
 - * max. owin
 - * RTT (also minimum, can we test it with ping?)
 - * hops (can we test it with traceroute?)
 - * percentage of TCP retransmissions (in bytes)
- Note that some of the above characteristics can be obtained from IPFIX records, some require per-packet monitoring and some require additional tests (by ping or traceroute)
- It may be interesting to visualize also difference values between two specified time points, such as the increase of traffic per country over the last year

3.5 Monitoring Wikipedia traffic

In a project that is related to Lobster, the Vrije Universiteit has been asked to monitor the traffic of the Amsterdam site of wikipedia, the community-based online encyclopedia. More concretely, the wikipedia administrators have absolutely no idea which pages are the most popular, and they really want to know. Monitoring this site is a challenge for three reasons:

1. The volume of the traffic is known to be large: in Amsterdam the load is spread over 11 servers.

2. The wikipedia system administrators want absolutely no interference with the current setup of the system. For performance and privacy reasons these systems do not produce any logfiles, and the administrators do not want to change this for this experiment.
3. The privacy of the users must not be compromised. Editing or even simply accessing wikipedia may be politically sensitive in some countries, and the wikipedia administrators want to be absolutely certain that any network monitoring does not reveal the identity of people that visit the site.

Fortunately these challenges can be met in our setup: to gather site access statistics we only need part of the url string for access; we don't need any information about the sender of the url request. We know how to handle this volume of information with Ruler, the packet processing tool developed in Lobster. By applying Ruler we can anonymise the monitored traffic as soon as possible, and at the same time we also reduce the volume of information that must be processed. Since we can use passive monitoring to collect the information, no interference with the current setup is necessary.

The current plan is to deploy a single PC with an IXP2350 network processing card: the initial filtering, anonymisation, and data reduction will be done by Ruler running on the network processor. Further data processing and recording to disk will be done by the host machine.

3.6 Per-application Network Traffic Characterization and Throttling

3.6.1 Overview

In the context of the LOBSTER project, we have implemented *Appmon*, a passive monitoring application for per-application network traffic classification. *Appmon* uses deep packet inspection to accurately attribute traffic flows to the applications that generate them, and reports in real time the network traffic breakdown through a Web-based GUI.

Appmon uses three different approaches for attributing flows to the applications that generate them. First, it searches inside application messages for characteristic application protocol patterns. For certain applications that dynamically negotiate the ports that are going to be used, *appmon* fully decodes the applications protocol to identify the new, dynamically generated port number and then tracks further traffic flows through these ports. Finally, legacy applications that do not match above filters are categorized based on well-known port numbers and protocols using BPF filters.

3.6.2 Extending Appmon for Per-application Traffic Throttling

One of the most frequent requests of network administrators is to identify the applications and hosts that generate the largest amount of network traffic. The emergence of peer-to-peer file sharing, multimedia streaming, and conferencing applications has resulted to a substantial increase in the traffic volume, since they transfer a large amount of data. Traditionally, traffic attribution to the corresponding applications is performed using the statically assigned port numbers. Widely used network services, like the Web, Telnet, SSH, and many others, are associated with well-known port numbers which can be used for identifying the traffic related with each application. Based on the corresponding port number, network administrators can give higher priority to the traffic of certain services, or allocated a limited number of resources for other applications, e.g. throttling highly bandwidth-consuming applications.

However, traffic management based on port numbers is becoming increasingly difficult, since many emerging applications, including popular bandwidth-hungry file sharing applications, do not use

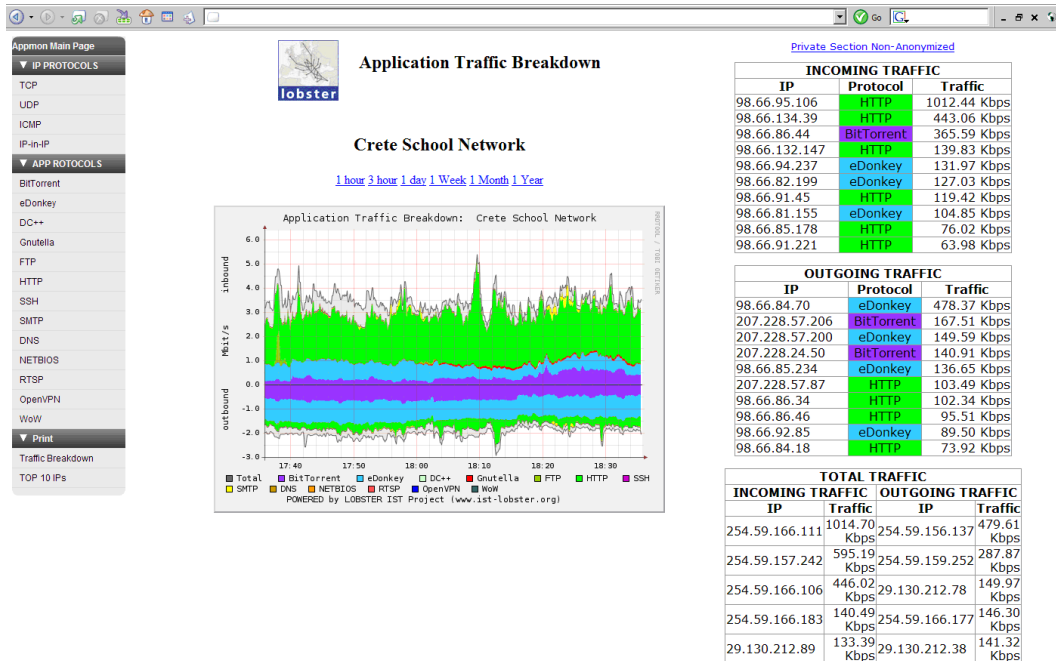


Figure 3.1: Appmon Web interface.

well-known port numbers. Instead, they allocate and use dynamically negotiated ports. Furthermore, some applications masquerade their traffic using pervasive, firewall-friendly protocols, like HTTP, in order to bypass firewall restrictions and make the identification of their traffic harder. Indeed, several widely used file sharing applications like BitTorrent can be configured to operate through port 80, which is usually left open even in environments with strict firewall configurations.

Appmon effectively identifies and tracks the traffic of such applications, and accurately determines the per-application network usage, even for applications that use dynamic ports or masquerade their traffic. Furthermore, Appmon characterizes the traffic of each network flow *separately*, which enables it to report the top bandwidth consuming IP addresses. Figure 3.1 shows the Appmon's Web interface, with the top-10 bandwidth consuming IP addresses on its right part.

This capability can be used for identifying misbehaving end-hosts that generate a significant amount traffic using a certain file sharing applications, which then can be throttled by configuring appropriately their gateway. Currently, this process can be performed only manually, i.e., identifying the top hosts with the highest bandwidth consumption through Appmon's web interface, and then updating the configuration of the corresponding router. In the future, we plan to extend Appmon with a module for dynamic router re-configuration in order to enable throttling end-hosts with exceptionally high bandwidth consumption due to non-productive file sharing applications.

3.7 Polymorphic Attack Detection

3.7.1 Introduction

The primary aim of an attacker or an Internet worm is to gain complete control over a target system. This is usually achieved by exploiting a vulnerability in a service running on the target system that

allows the attacker to divert its flow of control and execute arbitrary code, which is usually provided as part of the attack vector. Although the typical action of the injected code is to spawn a shell (hereby dubbed *shellcode*), the attacker can structure it to perform arbitrary actions under the privileges of the service that is being exploited. For example, the “shellcode” of recent worms usually just connects back to the previous victim, downloads the main body of the worm, and executes it.

Significant progress has been made in recent years towards detecting previously unknown code injection attacks at the network level using automated signature generation [3, 11, 4]. However, as organizations start deploying state-of-the-art detection technology, attackers are likely to react by employing advanced evasion techniques, such as polymorphism and metamorphism, known from the virus scene since the early 1990s [12], to defeat these defenses.

Polymorphic shellcode engines create different forms of the same initial shellcode by encrypting its body with a different random key each time, and by prepending to it a decryption routine that makes it self-decrypting. Since the decryptor itself cannot be encrypted, some intrusion detection systems rely on the identification of the decryption routine of polymorphic shellcodes. While naive encryption engines produce constant decryptor code, advanced polymorphic engines mutate the decryptor using metamorphism [13], which collectively refers to techniques such as dead-code insertion, code transposition, register reassignment, and instruction substitution, making the decryption routine difficult to fingerprint. While results have been promising, and some recent automated signature generation methods can cope with limited polymorphism [7, 6], most of the existing proposals can be easily defeated by attacks that employ extensive polymorphism and metamorphism.

3.7.2 Polymorphic Shellcode Detection using Emulation

Having identified the limitations of signature-based approaches, recent research efforts have turned to static binary code analysis for the detection of previously unknown polymorphic code injection attacks at the network level [14, 1, 8, 5, 2, 15]. These approaches treat the input network stream as potential machine code and analyze it for signs of malicious behavior. The first step of the analysis involves the decoding of the binary machine instructions into their corresponding assembly language representation, a process called disassembly. Some methods rely solely to disassembly for identifying long instruction chains that may denote the existence of a NOP sled [14, 1] or shellcode [8]. After the code has been disassembled, some techniques derive further control or data flow information that is then used for discriminating between shellcode and benign data [5, 2, 15].

However, after the flow of control reaches the shellcode, the attacker has complete freedom to structure it in a complex way that can thwart attempts to statically analyze it. For instance, by employing *self-modifying* code, which modifies its own instructions dynamically at runtime, an attacker can construct an attack that will eventually execute instructions that do not appear in the initial attack code as seen on the wire, on which static analysis methods operate on.

Static binary code analysis would need to be able to compute the output of each instruction in order to extract the real control and data flow of the code that will be eventually executed. This observation motivated us to explore whether it is possible to detect such highly obfuscated shellcode by actually *executing* it, using only information available at the network level [10, 9]. In contrast to previous work, the proposed approach relies on a NIDS-embedded CPU emulator that executes every potential instruction sequence that appears in the inspected network traffic, and compares its execution profile against the behavior observed to be inherent to polymorphic shellcode. Preliminary results demonstrate that the proposed approach is more robust to runtime evasion techniques like self-modifying code, compared to previous proposals.

3.7.3 Integration with LOBSTER

The current prototype implementation of the emulation-based detector uses a custom IA-32 CPU emulator. Instruction set simulation has been implemented interpretively, with a typical fetch, decode, and execute cycle. For our prototype, we have implemented a subset of the IA-32 instruction set, including most of the general-purpose instructions. However, *all* instructions are fully decoded, and if during execution an unimplemented instruction is encountered, the emulator proceeds normally to the next instruction. The implemented subset suffices for the complete and correct execution of the decryption part of all tested shellcodes [10].

In our future work, we plan to port the detector on top of MAPI, in order to exploit the distributed monitoring capabilities offered by the LOBSTER infrastructure. The current custom emulator can be integrated into a MAPI function, e.g., a function named `EMULATE`, which will allow the versatile implementation of different network-level polymorphic attack detection schemes based on code emulation. Indeed, as for example the MAPI `STR_SEARCH` function can be applied to any MAPI network flow in order to search the monitored traffic for malicious packets that contain a particular attack string, in the same way the `EMULATE` function will be used to scan the traffic of any network flow for the presence of polymorphic shellcode.

For instance, a detector for the protection of a web server would be easily implemented by creating a MAPI network flow for capturing the traffic, applying the necessary `BPF_FILTER` function in order to filter out unnecessary traffic and keep only the web requests, applying the `COOKING` function for IP defragmentation and TCP stream reassembly, and finally, applying the `EMULATE` function for the inspection of each incoming web request using code emulation. Then, by exploiting the LOBSTER monitoring architecture, this emulation-based detector will be able to transparently run on top of different monitoring hardware, such as common Ethernet NICs or DAG cards, as well as on either local or remote monitoring sensors, using the distributed monitoring capabilities of MAPI.

Chapter 4

Summary

In this deliverable the existing LOBSTER applications were first presented. Most of these applications are today installed in production networks and based on the experience with these applications several new potential LOBSTER applications or potential extensions to existing applications were described.

These new applications demonstrates some of the capabilities of the LOBSTER framework and provides some ideas as to what kind of applications that can be expected in the future. The described applications include both general monitoring applications that provides detailed information about the usage of the network as well as security application that can help network operators investigate security incidents or detect new worms more effectively. Some of the described applications are research oriented and may or may not be implemented in the near future, while others are already in the planning stage of implementation.

The LOBSTER framework is a powerful and versatile framework that allows rapid development of new monitoring applications, so the described applications should only be considered as a small subset of potential future applications. As the deployment of the LOBSTER technology increase an increasing number of applications will most likely become available.

Bibliography

- [1] P. Akritidis, E. P. Markatos, M. Polychronakis, and K. Anagnostakis. STRIDE: Polymorphic sled detection through instruction sequence analysis. In *Proceedings of the 20th IFIP International Information Security Conference (IFIP/SEC)*, June 2005.
- [2] R. Chinchani and E. V. D. Berg. A fast static analysis approach to detect exploit code inside network flows. In *Proceedings of the International Symposium on Recent Advances in Intrusion Detection (RAID)*, Sept. 2005.
- [3] H.-A. Kim and B. Karp. Autograph: Toward automated, distributed worm signature detection. In *Proceedings of the 13th USENIX Security Symposium*, pages 271–286, 2004.
- [4] C. Kreibich and J. Crowcroft. Honeycomb – creating intrusion detection signatures using honeypots. In *Proceedings of the Second Workshop on Hot Topics in Networks (HotNets-II)*, Nov. 2003.
- [5] C. Kruegel, E. Kirda, D. Mutz, W. Robertson, and G. Vigna. Polymorphic worm detection using structural information of executables. In *Proceedings of the International Symposium on Recent Advances in Intrusion Detection (RAID)*, Sept. 2005.
- [6] 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 2006 IEEE Symposium on Security and Privacy*, pages 32–47, 2006.
- [7] J. Newsome, B. Karp, and D. Song. Polygraph: Automatically generating signatures for polymorphic worms. In *Proceedings of the IEEE Security & Privacy Symposium*, pages 226–241, May 2005.
- [8] U. Payer, P. Teufl, and M. Lamberger. Hybrid engine for polymorphic shellcode detection. In *Proceedings of the Conference on Detection of Intrusions and Malware & Vulnerability Assessment (DIMVA)*, pages 19–31, July 2005.
- [9] M. Polychronakis, E. P. Markatos, and K. G. Anagnostakis. Network-level polymorphic shellcode detection using emulation. *Journal in Computer Virology*. To appear.
- [10] M. Polychronakis, E. P. Markatos, and K. G. Anagnostakis. Network-level polymorphic shellcode detection using emulation. In *Proceedings of the Third Conference on Detection of Intrusions and Malware & Vulnerability Assessment (DIMVA)*, pages 54–73, July 2006.
- [11] S. Singh, C. Estan, G. Varghese, and S. Savage. Automated worm fingerprinting. In *Proceedings of the 6th Symposium on Operating Systems Design & Implementation (OSDI)*, Dec. 2004.

BIBLIOGRAPHY

- [12] P. Ször. *The Art of Computer Virus Research and Defense*. Addison-Wesley Professional, February 2005.
- [13] P. Ször and P. Ferrie. Hunting for metamorphic. In *Proceedings of the Virus Bulletin Conference*, pages 123–144, Sept. 2001.
- [14] T. Toth and C. Kruegel. Accurate buffer overflow detection via abstract payload execution. In *Proceedings of the 5th Symposium on Recent Advances in Intrusion Detection (RAID)*, Oct. 2002.
- [15] X. Wang, C.-C. Pan, P. Liu, and S. Zhu. Sigfree: A signature-free buffer overflow attack blocker. In *Proceedings of the USENIX Security Symposium*, Aug. 2006.