VENO2 : a generic overlay infrastructure over heterogeneous networks

Yu, Kuan

2009

Yu, K. (2009). VENO2 : a generic overlay infrastructure over heterogeneous networks.
Doctoral thesis, Nanyang Technological University, Singapore.

<https://hdl.handle.net/10356/20727>

<https://doi.org/10.32657/10356/20727>

**NANYANG
TECHNOLOGICAL
UNIVERSITY**

DOCTOR OF PHILOSOPHY

**VENO2: A GENERIC OVERLAY INFRASTRUCTURE
OVER HETEROGENEOUS NETWORKS**

YU KUAN

YU KUAN

SCHOOL OF COMPUTER ENGINEERING

2009

2009

VENO2: A GENERIC OVERLAY INFRASTRUCTURE OVER HETEROGENEOUS NETWORKS

YU KUAN

School of Computer Engineering

A thesis submitted to the Nanyang Technological University
in fulfillment of the requirement for the degree of
Doctor of Philosophy

2009

Acknowledgments

I would like to take this opportunity to send out my appreciations to the people who influenced me and helped me complete this thesis.

First and foremost, I would like to thank my supervisor, Prof. Fu ChengPeng, for taking me as his student, and leading me to involve into this exciting research area. His perfect writing skills, his creative idea and quick thinking impress me deeply. Not only did he give me invaluable suggestions and timely support on my research, he was always willing to discuss with me no matter how strange my questions are. I would like to thank him for his patience and efforts to show me the way of doing research.

I benefited greatly from my group research partners: Zhou Bin, Zhang Ke, Zhang JianNing, Lu Xiao, Ma Yi, Zhang ZhunLei, Lv Qian, and Zou ZiXuan, for their invaluable simulations, experiments and happy discussions. Without them, this thesis would not be possible. Also, I would like to thank FYP students, Zhuo ZiQiang, Nguyen Tat Thang, Seng Zhe, and Kelvin Lee, for their assistance of programming and suggesting.

I sincerely express my thanks to Prof. Lau Chiew Tong, Prof. Foh Chuan Heng and Prof. Cai JianFei for serving on my qualifying exam and giving me valuable suggestions on how to conduct research. I am grateful to Prof. Lee Bu Sung, Prof. Jussipekka Leiwo and Prof. Sabu Emmanuel for their great effort of helping me clarify my research topic.

Special thanks to my lab technician, Linda, for her efficient technical support on solving problems of computer hardware and software. I would like to thank my colleagues at 4G

lab and CEMNET lab, past and present, for their kindly help and suggestions. Thanks also go to Nanyang Technological University and School of Computer Engineering for the financial support and research facilities.

I owe a special debt of gratitude to my parents and family. Words cannot express my gratitude to my parents, my brothers and my sisters who give me their support, love and encouragement from across the country. They instilled in me the value of hard work and taught me how to overcome life's disappointments.

Contents

Acknowledgments	i
List of Figures.....	vi
List of Tables.....	viii
Summary	9
Chapter 1 Introduction.....	11
1.1 Research Background	11
1.2 Problem Statements and Motivations	14
1.3 Summary of Contributions.....	15
1.3.1 Veno2 Infrastructure.....	15
1.3.2 VenoSec: Instance of Veno2 for Security Platform.....	16
1.3.3 VenoSink: Instance of Veno2 for IPv4/IPv6 Transition Platform	17
1.4 Structure of the Thesis	17
Chapter 2 Literature Review	19
2.1 Network Security	19
2.1.1 Review of Security Mechanisms	20
2.1.2 Evaluation of Security Mechanisms	29
2.2 IPv4/IPv6 Connectivity.....	34
2.2.1 Review of Transition Mechanisms.....	35
2.2.2 Evaluation of Transition Mechanisms	41
Chapter 3 Veno2 Architecture Proposal.....	47
3.1 Idea Proposal.....	47
3.2 Design Principles in Veno2 Agent	50
3.2.1 Transparency through Code Interposition.....	51

3.2.2	Extensibility and Flexibility by STREAMS Framework.....	52
3.2.3	Application-dependent Customization.....	52
3.3	Design Principles in Veno2 Gateway.....	53
3.3.1	Gateway Discover.....	54
3.3.2	Efficiency and reliability.....	54
Chapter 4	Veno2 Implementation.....	56
4.1	Overview.....	56
4.2	API Interception.....	58
4.2.1	Windows API Interception.....	58
4.2.2	Linux/Unix API Interception.....	60
4.2.3	Applying API Interception in Veno2 agent.....	61
4.3	STREAMS Framework.....	63
4.3.1	Architecture.....	64
4.4	Application Transformer.....	70
4.4.1	Design Goals.....	70
4.4.2	Implementation.....	72
4.4.3	Example.....	76
Chapter 5	Two Instances of Veno2 Architecture.....	79
5.1	VenoSec: Instance of Veno2 for Security Platform.....	79
5.1.1	Overview.....	79
5.1.2	Network Attacks and Security Services Model.....	80
5.1.3	VenoSec Mechanism.....	84
5.1.4	Technical Design.....	91
5.1.5	Experiment and Analysis.....	102
5.2	VenoSink: Instance of Veno2 for IPv4/IPv6 Transition Platform.....	119
5.2.1	Overview.....	119
5.2.2	VenoSink Mechanism.....	120
5.2.3	Technical Design.....	123
5.2.4	Experiment and Evaluation.....	129
Chapter 6	Conclusion and Future Work.....	136
6.1	Summary.....	136

6.2 Future Work 138

References 140

List of Figures

Figure 2.1 Application Specific Mechanism.....	21
Figure 2.2 SSL Layering	22
Figure 2.3 SSL is not transparent.....	23
Figure 2.4 IPSec Layering	24
Figure 2.5 Incompatibility between IPSec and NAT	25
Figure 2.6 SOCKS Layering.....	26
Figure 2.7 TESLA Mechanism	28
Figure 2.8 PES Mechanism.....	29
Figure 2.9 Dual Stack Mechanism	35
Figure 2.10 Tunneling Mechanism	36
Figure 2.11 NAT-PT	38
Figure 2.12 IVL.....	39
Figure 2.13 TRT	40
Figure 2.14 SOCKS64	40
Figure 3.1 Veno2 Architecture	47
Figure 4.1 Veno2 agent components	56
Figure 4.2 Key techniques in VenoSec agent.....	57
Figure 4.3 Invoking Loop Problem.....	62
Figure 4.4 Architecture of STREAMS Framework	64
Figure 4.5 STREAMS message structure	67
Figure 4.6 Architecture of Application Transformer.....	72
Figure 4.7 An example MSN message.....	73
Figure 4.8 Example code of Application Transformer for protecting MSN	77
Figure 5.1 VenoSec mechanisms for client-server application	84

Figure 5.2 VenoSec mechanisms for client-server-client application.....	86
Figure 5.3 Deploy in individual user scenario	88
Figure 5.4 Group user scenario without Veno2 gateway	89
Figure 5.5 Group user scenario with Veno2 gateway	90
Figure 5.6 Architecture of Veno2 agent for VenoSec.....	92
Figure 5.7 Using Application Transformer to protect MSN communication	94
Figure 5.8 Secure forwarding through Veno2 gateway.....	101
Figure 5.9 Applying VenoSec in two communication models.....	105
Figure 5.10 Applying VenoSec in three network scenarios	105
Figure 5.11 VenoSec works like a keyhole	107
Figure 5.12 VenoSec bridge the gap between security techniques and applications	108
Figure 5.13 Applying VenoSec for strong authentication	110
Figure 5.14 Applying VenoSec for confidentiality and integrity	112
Figure 5.15 Weaknesses	113
Figure 5.16 Testing environments.....	114
Figure 5.17 Veno2 bandwidth results in local host	115
Figure 5.18 Veno2 bandwidth results in LAN and WAN	116
Figure 5.19 Veno2 latency results	116
Figure 5.20 VenoSec bandwidth results in local host	117
Figure 5.21 VenoSec bandwidth results in LAN.....	118
Figure 5.22 VenoSec latency results	118
Figure 5.23 An Enterprise Scenario with IPv4 network and IPv6 network included	121
Figure 5.24 Architecture of Veno2 agent for VenoSink	123
Figure 5.25 Direct Forwarding by Data Forwarder	128
Figure 5.26 Four testing scenarios	131
Figure 5.27 VenoSink bandwidth results in LAN	132
Figure 5.28 VenoSink latency results in LAN	133
Figure 5.29 VenoSink bandwidth results in WAN	134
Figure 5.30 VenoSink latency results in WAN	134
Figure 6.1 Architecture of Universal Transport Platform	139

List of Tables

Table 2-1 Comparison of Security Mechanisms	33
Table 2-2 Comparison of Transition Mechanisms	46
Table 4-1 Message types	68
Table 5-1 Relationship between network attacks, services and techniques	83
Table 5-2 Example Policy Database	96
Table 5-3 Recommended security techniques	109

Summary

Today, the Internet has obtained an incredible success. But some aspects of the current Internet are limiting its further development to meet both current expectations as a trusted communication infrastructure and future demands for scaling to support a huge number of new users and devices. Security and IPv4/IPv6 connectivity are two mainly concerned problems among all of these aspects.

In order to tackle these problems, many security techniques and IPv4/IPv6 transition mechanisms have been proposed in recent years. However, as these proposals were usually patchworks intended to satisfy a particular requirement, they breach the coherence of the original Internet architecture, and some of them even conflict with each other. Considering engineering difficulties and practical deployment issues, these proposals are not widely used.

This thesis proposes a novel overlay infrastructure called Veno2, which aims to overcome critical deficiencies of the current Internet. Specifically, it is a newly designed open platform able to provide security, IPv4/IPv6 connectivity, mobility and performance and even other service functions in future. The salient feature of such Veno2 infrastructure is its great ease for deployment. There are no changes required on existing Internet infrastructures. Veno2 is implemented by using a STREAM framework.

Based on this Veno2 architecture, a security platform (called VenoSec) and an IPv4/IPv6 transition platform (called VenoSink), are derived respectively. VenoSec platform can protect new emerging client-server-client communications as well as traditional

client-server applications. VenoSink platform provides a transitional architecture between IPv4 and IPv6 networks so that applications can communicate with each other no matter they locate on the IPv6 networks or IPv4 networks. Both of them can be deployed easily.

Extensive network testbed experiments and live Internet measurements have been conducted for evaluating both performance and functionality. The results indicate that there is little performance degradation when VenoSec or VenoSink equipped.

Chapter 1

Introduction

1.1 Research Background

The Internet began in 1969 as the ARPANET [1], a project funded by the Advanced Research Projects Agency (ARPA) of the U.S. Department of Defense. The original goal of the project was to interconnect university and government computers primarily for noncommercial research use. Hence, the ARPANET is initially designed for a benign and trustworthy environment, with limited number of users. As being increasingly popular and attracting more and more users, the ARPANET officially became the Internet and moved from a government research project to a commercial network in 1989. In the next ten years, with vast companies and institutes being interconnected, the Internet rapidly grew into a communication medium for worldwide citizenry. During this process, however, two fundamental problems of the Internet limit its further development: insecurity and address exhaustion.

The Internet was originally designed for a benign and trustworthy environment, with no or little consideration for security. This assumption is clearly no longer valid for today's Internet, where commercial activities and private communications are conducted by millions of people spanning the entire global. Since the first internet worm Morris was widely distributed in 1988 [9], a lot of virus, cheated online trading and other dishonest behaviors were frequently spreading over the Internet.

To tackle these security problems, many techniques and protocols were proposed by researchers in the last 20 years. One category of techniques protect a specific application by patching the correspondent protocol with security functionality in an ad-hoc fashion, i.e. SSH [10], SET [11] and DNSSEC [12]. These techniques lack the generality for protecting other or new emerging applications. In contrast, more generic techniques, such as IPSec [13], SSL [14] and SOCKS [15] are proposed to provide security services for all applications. In practice, however, IPSec's inherent incompatibility with NAT (Network Address Translation) [34] restricts its deployment [35]. SSL is only widely used for Web application due to its requirement of modifying application code [45]. Also, SOCKS only protects communication between client and SOCKS server, but not full end-to-end path. As a result, all these techniques are not widely deployed.

Another critical issue limiting the development of the Internet is address exhaustion. The initial version of Internet Protocol (IPv4) [2] provides an addressing capability of about 4 billion (2^{32}), which were assumed sufficient in the early stage of the Internet. However, such implicit assumption was no long valid with the unanticipated explosive growth of the Internet. According to a prediction by the Organization for Economic Cooperation and Development (OECD) [3], the remaining IP addresses will run out by 2011. In order to solve this urgent problem, IPv6 [4] has been proposed by Internet Engineering Task Force (IETF) in 1996, which uses 128-bit address instead of 32-bit IPv4 address.

Since it is impractical to replace the existing IPv4 protocol instantly and simultaneously throughout the Internet, IPv6 will coexist with IPv4 for a long time. During this coexisting period, however, IPv6 brings connectivity problems due to lack of backward compatibility with the current Internet using IPv4. In last eight years, many transition technologies, i.e., 6to4 [5], NAT-PT [6], TRT [7] and SOCKS64 [8] have been proposed by IETF to solve these connectivity problems, and also some of them are ever tested in testbed. However, all these piecewise techniques have not been deployed on a large scale because of their applicability of only specific scenarios. As a matter of fact, users or ISPs have no interest to adopt IPv6 because of no obvious benefits immediately from deploying IPv6,

furthermore, their daily use of IPv4 resource (i.e., MSN [32], QQ [33], Games, Email, WWW) may be more or less disrupted and make them quite inconvenient. That is the fundamental reason why today the number of IPv6 users are quite behind expected and IPv6 bandwidth is scarcely used all the time in past few years.

As the above security and connectivity issues are fundamentally rooted in the underlying architecture design of the early stage Internet, revolutionary solutions by redesigning a new clean-slate Internet architecture from ground up are also proposed by researches over the past eight years. ITU's NGN (Next Generation Network) [18] and IETF's NGI (Next Generation Internet) [19] have been explored and experimented in different countries. In contrast, instead of seeking revolutionary solutions like NGN and NGI, GENI (Global Environment for Network Innovations) [20][21] and AGNs (Assurable Global Networks) [22] were proposed respectively by the NSF in August of 2005 and DARPA (Defense Advanced Research Projects Agency) in December of 2006. Both programs aim to explore some evolutionary technologies to gradually change and upgrade the existing Internet infrastructure.

Most recently, overlay network has been proposed as a way to add new functionalities into the Internet evolutionarily [56][57]. Compared with traditional methods, overlays require no significant change in the existing IP infrastructure, and thus do not involve significant expenditures. As a result, overlay network quickly becomes popular as a new entrant to the Internet market, and has been used for many applications, such as content distribution [58][59][60], routing [55][61], QoS [62][63], multicast [64][65], security [66] and bridging multiple address spaces [67]. Moreover, as observed in [57], overlays are not restricted to offering just isolated services, but also can evolve the Internet by introducing architectural innovations leading to tomorrow's Internet.

1.2 Problem Statements and Motivations

In recent years, the Internet continuously expands with a rate of 200 million more users per year. At the same time, as the end-users' desktop computers become increasingly powerful, many new client-centered applications rapidly emerge in the Internet and quickly become popular, such as P2P file sharing or video broadcasting, and grid-computing. These new applications together with the huge number of new users brought unforeseen stresses and strains that have revealed numerous limitations in the Internet architecture and triggered calls for new architectures to succeed it. Many Internet R&D projects have been proposed for revolutionary or evolutionary architectural change, such as NGN, NGI, GENI, AGN and overlay networks.

However, the ambitious goals of NGN and NGI for completely replacing the current Internet restrict them to be only in research community, primarily because of low user adoption and high deployment cost in practice. As a result, little progress have been made so far despite the huge amount of monies invested in IPv6 testbed by large countries like China, Japan, and the European Union. In addition, GENI only provides a set of guidelines, e.g. the most critical issues of improved security and network robustness should be resolved as soon as possible, while AGN openly acknowledges the lack of a viable solution and is seeking inputs from the public.

Also, most of current overlay networks do not target to provide a generic infrastructure, but only to address problems in a specific application area separately. Thus each of these overlays requires dedicated network resources (e.g. client agents or servers) to be deployed for each correspondent overlay network, which is usually redundant and sometimes even conflict. In addition, the general use of overlay is further hindered by tedious and complicated nature of porting legacy applications or equipping new applications with support for each particular overlay network.

In short, there currently does not exist any generic and practical solution to the root cause

of the today's Internet urgent challenges – security and transition.

1.3 Summary of Contributions

The contributions of this dissertation can be divided into three areas:

- ◆ Veno2 infrastructure is proposed to provide a generic architectural platform for offering extensible and flexible session layer services, such as security, IPv4/IPv6 transition, mobility, and performance enhancement.
- ◆ We make use of Veno2 infrastructure and create a security platform (VenoSec), which aims to provide various security services for all applications over the Internet. It is validated by real Internet experiments. One journal paper is being submitted.
- ◆ We make use of Veno2 infrastructure and create a transition platform (VenoSink) between IPv4 and IPv6 networks so that hosts in IPv4 networks can communicate with hosts on the IPv6 networks. It has also been validated by experiments in testbed and Internet. One journal paper is being submitted.

1.3.1 Veno2 Infrastructure

The main contribution of this thesis is a new generic overlay infrastructure over the Internet called Veno2. It can be quickly deployed over the Internet for providing extensible and flexible services, such as security, IPv4/IPv6 connectivity and transition, mobility and performance enhancement.

Firstly, we propose Veno2 infrastructure and design it to be composed of Veno2 agent and Veno2 gateway. Veno2 agent is software module installed at end hosts and is responsible for complex processing, such as security process, and IPv4/IPv6 transition. Veno2

gateway is a gateway server to provide relative simple services, like data forwarding or encapsulating/decapsulating data securely.

Second, we implement the Veno2 infrastructure on both Windows and Linux platforms. Specifically, we integrate three key techniques: code interception technique for enabling transparent deployment of Veno2 agent, STREAMS framework for extensibility and flexibility when incorporating and configuring new services or protocols into the infrastructure, and generic application transformer for facilitating application-dependent customization.

Finally, we conduct experiments on both testbed and real Internet environments, and the results indicate that the performance penalty caused by Veno2 infrastructure is acceptable for practical use.

1.3.2 VenoSec: Instance of Veno2 for Security Platform

In the security area, we make use of Veno2 architecture to design and implement the VenoSec security platform. In particular, we propose practical mechanisms for deploying VenoSec in both individual and group user scenarios for either traditional client-server applications or new emerging client-server-client communications.

We also have implemented VenoSec with modular components, namely application transformer, security policy manager including a policy database and selector, and security provider interface that encapsulates and interfaces various security algorithms and packages, such as encryption/decryption, key exchange algorithms, Kerberos [36], and SSL packages [45].

At last, we evaluate the performance degradation incurred by VenoSec on testbed and real Internet, and demonstrate that the performance is acceptable for practical use. Moreover,

we employ VenoSec to protect various existing software, and the results validate its functionality.

1.3.3 VenoSink: Instance of Veno2 for IPv4/IPv6 Transition Platform

In this work, we make use of Veno2 architecture to design and implement the VenoSink transition platform to solve transition and connectivity problems between IPv4 and IPv6 networks. Specifically, we design the functionalities of Veno2 agent and Veno2 gateway in VenoSink platform and demonstrate its benefits when applied in different usage scenarios.

In addition, we implement VenoSink with innovative techniques, such as application transformer for transforming applications with embedded IPv4/IPv6 address.

Meanwhile, the experiments conducted on both testbed and real Internet show that the performance after employing VenoSink is acceptable for practical use. We also validate the functionality of VenoSink by demonstrating that it supports various popular user applications.

1.4 Structure of the Thesis

The structure of the thesis is as follows.

Chapter 2 reviews main techniques in network security and IPv4/IPv6 transition areas respectively. Section 2.1 classifies network security mechanisms into three types: application specific, application generic, and interception based. These security

mechanisms are then evaluated with five metrics: security services, applicability, transparency, compatibility and deployability. Similarly, Section 2.2 introduces popular IPv4/IPv6 transition mechanisms in three categories: dual stack, tunneling and translation, and evaluates them with metrics, namely robustness, compatibility, deployability, scalability and efficiency.

In Chapter 3, we propose the Veno2 infrastructure as a universal overlay platform for providing services like network security and IPv4/IPv6 transition. Section 3.1 presents the overview of the infrastructure. Then Section 3.2 and Section 3.3 describe the two basic components in the infrastructure: Veno2 agent and Veno2 gateway respectively, and discusses the key techniques employed by them.

Chapter 4 presents the fundamental implementation of Veno2 infrastructure. Specifically, we describe the implementation of three key techniques employed in the infrastructure: Section 4.2 explains the API interception techniques in both Windows and Linux/Unix systems and their usage in Veno2; Section 4.3 discusses about STREAMS framework; and Section 4.4 presents application transformer.

Chapter 5 contains experimental evaluation of two instances of Veno2 infrastructure: VenoSec (Section 5.1) and VenoSink (Section 5.2). For both of these two sub-architectures, we first describe their mechanisms when deployed in various environments; this is followed by technical framework presenting the actual implementation; and finally, we show experiment results obtained from both testbed and real Internet.

In Chapter 6, we conclude this thesis and give suggestions for future work.

Chapter 2

Literature Review

This chapter reviews main techniques in network security and IPv4/IPv6 transition areas respectively. Section 2.1 classifies network security mechanisms into three types: application specific, application generic, and interception based. These security mechanisms are then evaluated with five metrics: security services, applicability, transparency, compatibility and deployability. Similarly, Section 2.2 introduces popular IPv4/IPv6 transition mechanisms in three categories: dual stack, tunneling and translation, and evaluates them with metrics, namely robustness, compatibility, deployability, scalability and efficiency.

2.1 Network Security

The Internet was not created with security in mind and most of the techniques of communicating between computers do very little to protect that data [21]. Telnet, FTP, HTTP and even email all send data over the network in clear, unencrypted text. This causes some problems as anyone along the way can easily see what you are sending. This means someone can easily grab your username and password and then access your system (through your account). Once this happens, your system is no longer secure. To protect communication from this threat, many security techniques and mechanisms, i.e. SSH [10], HTTPS [16], SSL [14] and IPSec [13] have been proposed in past 30 years.

In this section, we review these different security mechanisms and evaluate them based on the main five metrics, namely, *security services*, *applicability*, *transparency*, *compatibility* and *deployability*. *Security Services* describe the services that can be provided by a security mechanism, i.e. authentication, confidentiality, integrity, access control, and etc; *applicability* depicts the range of applications that a security mechanism can support; *transparency* here means the ability to enhance security of an application without relinking or recompiling the correspondent client or server software; *compatibility* implies that the new protocol must be able to interoperate with existing protocols, and the interoperation must not negatively affect one party; at last, *deployability* means that a protocol can be easily integrated into the existing network infrastructure, without requiring fundamental or many modifications to existing protocols or network hardware.

The following sections present a detailed review about the security mechanisms and indicate the obstacle of them. Section 2.1.1 introduces the popular security mechanisms as well as research works related to our proposal. Section 2.1.2 evaluates these security mechanisms from the 5 metrics talked above and discusses the obstacle in real deployment.

2.1.1 Review of Security Mechanisms

In this section, we review the security mechanisms classified into three categories: *application specific*, *application generic*, and *interception based*. Meanwhile, the five main metrics are applied to evaluate each of these mechanisms analytically.

Application Specific Mechanism

Application specific mechanism is the most simple and straightforward way to protect an application, with which a correspondent secure protocol is designed and implemented for

each application. For example, as shown in Figure 2.1, SSH enhances security to Telnet, FTPS [43] protects FTP, HTTPS [16] is for HTTP, and POPS [44] makes POP (for email post office) secure.

Figure 2.1 Application Specific Mechanism

However, although the application specific mechanism is the most straightforward way to protect an application, it has several problems. First, the applicability is poor. Each protocol can only protect its correspondent application, but not others. Thus, each application requires a distinct secure protocol to protect it. But, in fact, most applications share some common security requirements, such as authentication, confidentiality and integrity. Therefore, a more efficient approach for offering security is to design one generic secure protocol, which can offer various security services and allow different applications to access these services according to their specific security requirements [21].

Deployability is the second problem of the application specific mechanism. Nowadays, a user's computer always supports a lot of applications. For instance, a user may use a computer to remote login through Telnet, transfer files through FTP, surf the Web through HTTP, and fetch email through POP. With the application specific mechanism, the user needs to deploy the correspondent secure software for each application to replace the old insecure one. As a result, the deployment is hindered by the extra work involved for installing, configuring and learning how to use the new secure software for each application.

Application Generic Mechanism

As discussed in above paragraphs, the problems of application specific mechanism results from the design approach of requiring a separate secure protocol for each application. To tackle these limitations, security researchers proposed application generic mechanism, which allows one security protocol/technique to universally provide security services for many applications. The essential idea is to integrate security into a lower layer in the network protocol stack. By protecting this lower layer, applications running above it are also automatically protected. Typical examples are SSL, IPsec and SOCKS5.

SSL (Secure Socket Layer)

As shown in Figure 2.2, SSL (Secure Socket Layer) [14] runs above TCP/IP and below higher-level application protocols. It works by cryptographically transforming packets from higher application layer prior to passing them to lower TCP/IP processing. The set of security services provided by SSL include mutual authentication, integrity and confidentiality.

Figure 2.2 SSL Layering

Observing the figure, SSL consists of two main sub-protocols: Handshake Protocol and Record Protocol. The Handshake Protocol allows the client and server to authenticate each other and to negotiate cryptography algorithms and keys before transmitting application data. The Record Protocol is responsible to cryptographically encapsulate the data from various higher-level applications. This encapsulation offers confidentiality through encrypting/decrypting the data with a symmetric key, and offers integrity by including an integrity check with a keyed MAC algorithm. The keys and algorithms are

previously negotiated by the Handshake Protocol.

In theory, since SSL works below application layer, it is able to support all TCP/IP applications. Hence, SSL provides good applicability.

However, SSL does not provide transparency because application software needs to be modified and updated in order to integrate with SSL. Specifically, SSL is implemented above the socket layer, which defines the Application Programming Interface (API) for network programming. As depicted in Figure 2.3, application software invokes the standard routines supplied by the socket layer to perform network communication. For example, to send data, the software calls the *send()* function, which is then responsible to invoke lower layer TCP/IP modules to actually send out the data.

Figure 2.3 SSL is not transparent

To provide security services, SSL defines an alternative API for network programming, which runs above the standard API in the socket layer, seeing the figure. As a result, for application software to access those security services, it must invoke the routines in the new API defined by SSL. For example, to send data securely, the software must call the *SSL_send()* function instead of the standard *send()* function. Consequently, application software must be modified by replacing original socket API with correspondent SSL API in order to employ SSL [45].

Because of the above non-transparency problem, users are required to deploy special client or server software with SSL support in their computers. As a result, the cost for

installing, configuring and learning the new secure software for each application inhibits SSL from widely deployment for various applications. In practice, it is only widely implemented for HTTP communication. This is because SSL support is built into most Web client browsers and Web servers. But there is usually no such built-in SSL support in the software for most other applications.

IPSec (IP Security)

IPSec (IP Security) [13] is a collective name for the security architecture, which is designed to provide various security services to the network traffic at the IP layer, in both IPv4 and IPv6 environments (optional for IPv4 and mandatory for IPv6). These security services include access control, data integrity check, data origin authentication, replay protection and confidentiality. All the component protocols in IPSec have been standardized by the IETF (Internet Engineering Task Force). IPSec is mainly employed for implementing VPN (Virtual Private Network).

Figure 2.4 IPSec Layering

As shown in Figure 2.4, IPSec consists of three main component protocols: IKE (The Internet Key Exchange), AH (Authentication Header) and ESP (Encapsulating Security Payload). IKE is an application layer protocol that authenticates the peer ends and negotiates cryptography algorithms and keys to be used by AH and ESP. Both AH and ESP works in network layer. Essentially, each IP packet is transformed into a correspondent AH or ESP packet that contains security information such as authentication data, integrity check, and optionally encrypted payload. AH provides integrity,

authentication and anti-replay protection, while ESP can additionally provides confidentiality through encryption/decryption.

Because IPSec operates in network layer, it secures all data transmitted between end points regardless of application. Hence IPSec has good applicability. Besides, IPSec does not require modifying or updating the client or server application software. Therefore, it also provides transparency.

The main problem of IPSec is its incompatibility with the NAT (Network Address Translation) [34] technique that is widely deployed in today's Internet. Essentially, IPSec is designed to protect not only payload of IP packet, but also IP headers. Figure 2.5 briefly shows how IPSec protects IP header from being modified during transmission. Suppose Alice sends a packet to Bob. When the packet reaches Alice's IPSec module, IPSec inserts some cryptography information into the packet. And before Bob obtains the packet, his IPSec module checks the cryptography information to decide whether the IP header changes or not. If it changes, an attacker may modify the IP header, thus the packet is considered as insecure and is then discarded [35].

Figure 2.5 Incompatibility between IPSec and NAT

NAT works by modifying IP address information in the header of an IP packet, causing IPSec to deem the packet as insecure and discard it mistakenly. As illustrated in Figure 2.5, after Alice has sent out a packet and her IPSec module has processed it, NAT

modifies the IP header. Then, before Bob obtains the packet, his IPSec module checks the packet to find that it has been changed, and thus discards it by mistake. As a result, this IPSec-NAT incompatibility has become a major obstacle in the deployment of IPSec.

In addition, IPSec needs to insert its processing module into the kernel of a computer's operating system. Consequently, the IPSec client software can only be deployed by users who have administrative privilege of a computer. This limitation makes it difficult to extend IPSec deployment in public environments, such as public computers in shopping center, library or airport, where users do not have administrative privilege.

SOCKS

The SOCKS [15] protocol is essentially a proxy protocol, which can additionally provide security services. As depicted in Figure 2.6, SOCKS consists of two components, the SOCKS server and the SOCKS client. The former is implemented at the application layer in an intermediate host between the application client and server, and the latter is conceptually a "shim-layer" between the application layer and the transport layer. Specifically, any data from application client is intercepted by the SOCKS client, which encrypts the data and securely relayed to the SOCKS server. The SOCKS server decrypts the data and in turn delivers the original application data to the application server.

Figure 2.6 SOCKS Layering

Because SOCKS protocol operates below application layer, it can protect all applications, and hence has good applicability. In addition, SOCKS also provides transparency with the help of socksifier – special SOCKS client software like SocksCap [49] that can divert any application data to go through SOCKS server without modifying the existing

application software.

However, the security services provided by SOCKS are quite limited for two reasons. First, the SOCKS protocol is basically designed for proxy, which occurs mainly between the application client and the SOCKS server. As a result, no security is guaranteed between the SOCKS server and the application server. An attacker can easily mount attacks by monitoring the data traffic between the SOCKS server and destination.

Moreover, the security mechanisms employed by SOCKS are either too weak (e.g. plaintext username/Password authentication without encryption) [47] or too complicated (e.g. GSS-API authentication using Kerberos) [48]. As a result, currently SOCKS is mainly used for proxy purpose, typically without or with very weak security protection.

Interception Based Mechanism

Here, interception based security mechanism implies security techniques that enforce security by intercepting network operations performed by application software. For example, the SocksCap [49] software can intercept socket APIs from application software, and replace these socket APIs with correspondent secure ones by incorporating security implementation into them. In the following paragraphs, we discuss two interception based security mechanisms: TESLA and PES.

TESLA (Transparent, Extensible Session-Layer Architecture)

TESLA (invented by MIT 2002) [50] attempts to transparently provide certain popular session layer services, such as encryption, compression, SOCKS, flow mitigation and etc. The motivation behind TESLA is to free application software developers from the intricate process of implementing these popular services.

As illustrated in Figure 2.7, TESLA provides an interposition (or “shim”) layer between applications and TCP/IP modules in OS kernel. Similar to SOCKS, this interposition is

transparent by the use of dynamic library technique, which consequently avoids making changes to either operation system or applications.

Figure 2.7 TESLA Mechanism

Furthermore, each session service is implemented as an instantiation of a flow handler, and TESLA combines these handlers according to user's requirement. For example, with the reference to Figure 2.7, if a user expects to compress and then encrypt data before sends them out, TESLA can provide these services by inserting the compression and encryption flow handlers into the shim layer in a chained manner. Of course, to decrypt and decompress the data, the server must also invoke corresponding flow handlers.

TESLA facilitates providing session layer services in a generic fashion. However, it offers little assistance in providing security services in particular. For instance, although TESLA can apply encryption flow handler to encrypt data, it neither specifies how to obtain the secret key for encryption, nor includes any mechanism for authentication, which is necessary in network security. Moreover, the current implementation for TESLA is only for Linux/Unix, hence it is impossible to deploy it in widely used Windows platform.

PES (Privacy Enhanced Sockets)

PES (Privacy Enhanced Sockets) [51][52] provides a user-level subsystem to protect users' privacy in network communication. Conceptually, PES can be considered as a SOCKS proxy system, except that both SOCKS server and application server co-reside at

a same target computer.

Figure 2.8 PES Mechanism

Figure 2.8 illustrates two alternative architectures of PES. On the client side, PES works similar to SOCKS or TESLA by interposing itself between applications and TCP/IP modules transparently. On the server side, a PES proxy is responsible to establish secure channel with the client PES library. After the secure channel is successfully established, it is handled on the server side either by the application server, if it explicitly links with the PES library (alternative 1), or otherwise by the PES proxy (alternative 2).

PES provides end-to-end authentication, confidentiality and integrity services to the application being protected. However, PES is only for pure client-server network communication, not considering client-server-client communications, in which two clients communicate with each other through one or more intermediate servers, such as MSN [32], ICQ [53], email, and etc. For these applications, PES will fail because it encrypts all the application data causing the intermediate server unable to recognize the encrypted data. Moreover, the end-to-end security services provided by PES are difficult to be applied to complicated network scenarios, such as communication between a user and his organization, or between two organizations, where network topology and user management should also be considered.

2.1.2 Evaluation of Security Mechanisms

From the review of various security mechanisms above, we find that each mechanism has its own advantages and disadvantages. However, to help assess them qualified for practical use or not, we apply following five main criteria for evaluation: *security services, applicability, transparency, compatibility* and *deployability*.

Security Services

Security services indicate what kind of protection is provided by a network security mechanism. For example, authentication service allows two communication entities to confirm each other's identity, confidentiality or privacy protects sensitive information from being disclosed by unintended entity, and integrity service makes sure that data is not modified in transit.

Application specific mechanisms, such as SSH, FTPS, and POPS, are usually closely related to the correspondent application and offer thorough security services for protecting the application. Meanwhile, SSL and IPSec provide configurable security services to allow protecting various applications according to their specific security requirements.

However, SOCKS mechanism is not originally designed for security purpose, and the security services offered by it are either too weak (plaintext username/password authentication without encrypting data) or too complicated (GSS-API implementation using Kerberos). TESLA is also not dedicatedly designed for network security, and it does not indicate what security services are provided. It neither specifies how to obtain the secret key for encryption, nor includes any mechanism for authentication, which is necessary in network security..

Applicability

Applicability depicts the range of applications that a security mechanism can support. Because the designer or software programmer of an application may not be a security

expert at the same time, a generic security mechanism allows them to easily incorporate security functionalities into the application without understanding the underlying details of security protocols and algorithms.

For the application-specific mechanism, each secure application protocol only protects its correspondent application, but not others, hence it has poor applicability.

To some extent, all the other mechanisms provide good applicability since they all support various applications. However, some of them cannot work for certain types of applications and are thus not as applicable as other security mechanisms. For example, SSL offers security service for just TCP but not UDP traffic, so it cannot protect applications based on UDP, such as VoIP, game and video applications. SOCKS mechanism is designed for protecting only client applications, and thus is not suitable to be employed in server applications. Also, TESLA does not provide dedicated security modules, so it is difficult to apply TESLA for protecting various applications with distinct security requirements. In addition, PES does not support client-server-client applications which are increasingly popular in nowadays Internet.

Transparency

Transparency here means the ability to enhance security of an application without relinking or recompiling the correspondent client or server software. In today's Internet, there are many popular legacy applications that have no security support, and also many new applications are still not developed with security in mind. The transparency feather helps to integrate security into these insecure applications without redeveloping the existing client or server software, which usually involves high cost and long period.

The application specific mechanism requires correspondent secure software to replace the old insecure one for each application. Thus it has poor transparency. Although SSL claims to support various applications, it requires the application software to be relinked with SSL library. Therefore SSL still needs to modify client or server software, and hence

is not transparent.

The other security mechanisms provide good transparency since they all do not require modifying client or server software for any application.

Compatibility

Compatible here implies that the new protocol must be able to interoperate with existing protocols, and the interoperation must not negatively affect one party. Of course, the old protocol must also be compatible with the new protocol.

Among the mentioned security mechanisms, IPSec is the only one that has poor compatibility. With the popular using of NAT in nowadays Internet, the incompatibility between IPSec and NAT has become major obstacle in wide deployment of IPSec.

Deployability

Deployability means that a protocol can be easily integrated into the existing network infrastructure, without requiring fundamental or many modifications to existing protocols or network hardware.

Actually, deployability is related to both transparency and compatibility. If a security mechanism has poor transparency, users need to deploy correspondent secure software in their own computers for each application. Considering the great number of applications in a user's computer today, the deployment is hindered by additional work for installing, configuring and learning to use the new secure software, especially for non-technical users, such as marketing and sales people and executives. Thus, the application specific mechanism and SSL are not equipped with good deployability since they are not transparent. Also, IPSec is not widely deployed because it is not compatible with the popular employed technique – NAT.

Moreover, SOCKS mechanism is designed for proxy usage, which requires a SOCKS server to be deployed between the application client and server. Thus the extra cost for installing and configuring the SOCKS server inhibits SOCKS mechanism from widely deployment. In addition, PES only considers end-to-end situation, which makes it difficult to be deployed in complicated network scenarios, such as communication between a user and his organization, or between two organizations.

From the comparison we made above, we draw a table to simply assess the applicability of each mechanism from the five aspects.

	Application specific	Application generic			Interception based	
		SSL	IPSec	SOCKS	TESLA	PES
Security Services	✓	✓	✓	✗	✗	✓
Applicability	✗	✗	✓	✗	✗	✗
Transparency	✗	✗	✓	✓	✓	✓
Compatibility	✓	✓	✗	✓	✓	✓
Deployability	✗	✗	✗	✗	✓	✗

Table 2-1 Comparison of Security Mechanisms

In Table 2-1, the five characteristics of every security mechanism are roughly depicted. Attention must be paid that in a good and complete protocol design, all the criteria must be considered. However, there are two issues that complicate the matter. First, all the criteria are not exactly measurable. For example, no explicit metrics can be used to calculate the applicability or deployability of a protocol. It is up to the protocol designer's intuition and insights to determine whether a certain criterion is adequately considered. Second, the criteria are correlated with each other most of the time. It is impossible to design a protocol that performs extremely well in all criteria; instead, some tradeoffs are often necessary. A certain criterion may need to be sacrificed in order to provide better performance in another criterion. Thus, besides considering each of the criteria, the

relationship between the criteria must be observed.

2.2 IPv4/IPv6 Connectivity

IPv6 [4] not only brings solution to the scarcity of IP address, but it also brings new problems due to lack of backward compatibility with the current Internet using IPv4. In last ten years, many transition technologies, i.e., 6to4, NAT-PT, TRT and SOCKS64 have been proposed by researches to solve these incompatibility problems, and also some of them are ever tested in testbed.

In this section, we give a review on these different transition technologies and compare each other using the main five metrics, namely, *robustness*, *compatibility*, *deployability*, *scalability* and *efficiency*. A *robust* protocol is defined to be able to function correctly, with a certain degree of tolerance, even though factors that influence its operations are constantly changing; *compatibility* here means that the new protocol must be able to interoperate with existing protocols, and the interoperation must not negatively affect one party; A protocol can be easily integrated into the existing network infrastructure, without requiring fundamental or many modifications to existing protocols or network hardware. This is called *deployability*; the *scalability* of a protocol implies how well the protocol handles increasing load or complexity; at last, the *efficiency* of a protocol means good performance, such as high throughput or small delay, without being excessively aggressive, but by making better use of available resources.

The following sections present a detailed review about the status of current transition mechanisms and indicate the current research's obstacle. Section 2.2.1 introduces the basic IPv6 transition mechanisms. Section 2.2.2 compares these transition mechanisms from the 5 metrics talked above and discusses the critical obstacle to widely deployment.

2.2.1 Review of Transition Mechanisms

In this section, we will review three main types of transition mechanisms: Dual Stack, Tunneling, and Translation. Meanwhile, the five main metrics are applied to evaluate each of these mechanisms analytically.

Dual Stack Mechanism

Dual Stack [28] is the most simple and straightforward way for IPv6 nodes to remain compatible with IPv4-only nodes. Meanwhile, it also provides the direct connectivity between two nodes. In Dual Stack mechanism, both IPv4 and IPv6 protocol stacks are deployed on the same node. And these nodes are thus called dual stack nodes. With the two protocols equipped, these Dual Stack nodes have the ability to send/receive both IPv4 and IPv6 packets.

Figure 2.9 Dual Stack Mechanism

However, although Dual Stack mechanism is the most straightforward way of transition, it has two main problems. First, each dual stack host or router has to keep two separate routing tables as both IPv4 and IPv6 must be equipped, which means more memory and CPU power consuming. Second, network application running on Dual Stack node is not able to work as usual in some conditions though the node equips both IP protocols. For example, two dual stack nodes in IPv4 network are not able to communicate with each other using IPv6 application. Therefore, Dual Stack mechanism only has limited

deployability and compatibility.

Tunneling Mechanism

During the long coexisting period, while the IPv6 network is being deployed, the existing IPv4 infrastructure can still remain functional. Accordingly, to utilize existing IPv4 infrastructure to carry IPv6 traffic, Tunneling [37] technology was proposed. With this technology, the tunnels provide virtual connectivity over the physical network.

In typical tunneling mechanisms, current IPv4 network is treated as a unicast virtual link layer, and current IPv4 routing infrastructure is used to forward IPv6-over-IPv4 encapsulation packets. The host in one IPv6 site uses a special IPv6 address (e.g. 2002:IPv4_address::/80) as communication identifier when IPv6 packet go through the border router and the end point of tunnel can be found within the special address. Finally, the border router in the other site decapsulates the packet, and forwards this packet to its real IPv6 destination. However, the routing problem should be meticulously considered, since the packets have to be forwarded to the border router to perform encapsulation/decapsulation. The commonly used tunneling mechanisms include 6to4 [5], ISATAP [38], Teredo [39], Tunnel broker [40], DSTM [41], etc.

Figure 2.10 Tunneling Mechanism

6to4 is a typical tunneling mechanism, so it equips both advantages and disadvantages of the tunnel technology. It is compatible with current IPv4 and scalable on a large site basis as it introduces no new IPv4 routing table entries and only one new entry into IPv6's

global routing table. However, single point failure is inevitable due to the border router's encapsulation/decapsulation function.

As one variant of tunneling technology, ISATAP (Intra-Site Automatic Tunnel Address Protocol) also uses IPv4 infrastructure to get IPv6 connectivity. However, does not like 6to4 operating between two IPv6 sites, ISATAP connects dual stack nodes isolated within IPv4 network. Yet ISATAP's scalability is not as good as 6to4 because it is particularly designed for host-to-host tunneling.

To simplify the routing problem in tunneling mechanisms, Tunnel Broker was presented. This mechanism makes use of a component named tunnel broker to replace meticulous routing design, for user to manually connect, register and activate tunnels to Tunnel Server. And it manages tunnel creations, modification and deletion on behalf of user. However, the overhead of manual configuration is a limit for its deployability. Furthermore, the use of the tunnel broker component introduces the single point failure.

Although most Tunneling mechanisms can achieve virtual connectivity between two IPv6-only nodes, they (e.g. 6to4, ISATAP) have problem in communicating with IPv6 host located behind IPv4 NAT. Teredo is just the mechanism brings IPv6 connectivity to IPv6 nodes located behind IPv4 NATs by using user datagram protocol (UDP). But this mechanism is just designed for bringing connectivity to user located behind IPv4 NAT. Thus, its low compatibility makes it not feasible for practical implementation.

The tunneling mechanisms talked above are all IPv6-over-IPv4 tunneling used in IPv4 dominant network. But we also need IPv4-over-IPv6 virtual connectivity in future's IPv6 dominant network. DSTM is designed for this purpose. However, since DSTM is a mechanism designed for future use, it is not that useful at current stage of transition.

Translation Mechanism

Despite some progress has been made by Dual Stack and Tunneling mechanisms in

certain scenario, neither of them focuses on the indirect connectivity/communication between IPv6-only node and IPv4-only node. Translation technology is thus designed for getting this indirect connectivity between two different networks. Specifically, translation between two networks can take place at either IP layer, transport layer, or application layer. In detail, NAT-PT [6], IVI [29], BIS [30] are IP layer translation mechanisms; transport layer mechanisms mainly include TRT [7]; application layer mechanisms involve BIA [31], SOCKS64 [8].

Figure 2.11 NAT-PT

NAT-PT refers to Network Address Translation-Protocol Translation, working at IP layer. It builds on IPv4 NAT device to provide indirect connectivity between IPv4 network and IPv6 network. In NAT-PT, translation is made between IPv4 and IPv6 protocol by a certain NAT-PT gateway. As sessions are initialized, this NAT-PT gateway dynamically assigns IPv4 addresses to IPv6-only nodes from an IPv4 address pool. By using the temporarily assigned IPv4 address, IPv6-only node in the IPv6 network is able to communicate with IPv4-only node in IPv4 network. NAT-PT could be regarded as a reasonably mature mechanism as it is based on existing IPv4 NAT. At the same time, due to the use of existing NAT device, NAT-PT not only suffers the problems common to other translation mechanisms like single point failure, it also inherits some from NAT, i.e. it is mandatory that all requests and responses pertaining to a session be routed via the same NAT-PT router, and it is not scalable enough for large-scale deployment due to keeping too much state information in gateway.

Figure 2.12 IVI

IVI is proposed to solve the scalability problem of NAT-PT by making translation completely stateless. Moreover, IVI introduces multiplex into the mapping to adapting the trend of IPv4 addresses being exhausting. Specifically, a prefix-specific and stateless address mapping scheme is proposed as shown in Figure 2.12. In particular, 72 bit to 79 bit is port divisor indicator (PDI) indicating how many IPv6 addresses sharing a same IPv4 address in the power of 2. 80 bit to 95 bit is port prefix indicator (PPI) used to indicate the port range a specific IPv6 host can use as a port prefix. With this address mapping scheme, IPv4/IPv6 addresses can be translated statelessly, and multiple IPv6 addresses can be mapped into a same IPv4 address, partitioned with non-overlapped port range. However, IVI requires end users to modify either their applications or OS's bind() function to recognize the address allocated to them, which bring deployability difficulty. Besides, to relax the IPv4 address utilization problem while keep translation stateless, IVI makes tradeoff by limiting port usage for end hosts, which may cause a duration of shortage for hosts who demand more port to establish connection.

BIS (Bump In the Stack) is also an IP layer translation mechanism, but it is to support dual stack hosts using IPv4 application to communicate with IPv6 hosts using IPv6 application without need of any dedicated server. This is achieved by three modules inserted between TCP/IPv4 and network card driver, making each host self-translate by adding segments to the IP stack. Therefore, BIS can be actually seen as a very extension of NAT-PT, using the same methods. But like NAT-PT, BIS also employs MTU, fragmentation problem and it does not support IPsec either. And it is worth pointing out that BIS needs to modify the kernel as it inserts three modules between TCP/IPv4 and network card driver. It is thus hard to deploy in practice.

Figure 2.13 TRT

TRT (Transport Relay Translator) is a transport layer translator enables IPv6-only host to exchange TCP and UDP packets with IPv4-only host by using a dedicated server named TRT system. It works at the border of two domains and acts like an intermediary. Since TRT works at transport layer, it is free from MTU and fragment problem. However, TRT is not a good deployable mechanism due to the unidirectional translation characteristic (from IPv6 to IPv4) and the need of special DNS server or modified DNS resolver (add prefix to resolved IPv4 address) implementation on initiating host.

Figure 2.14 SOCKS64

SOCKS64 (SOCKS-based IPv6/IPv4 Gateway Mechanism) is an application layer translation tool. It relays a flow between IPv4 and IPv6 hosts through a dedicated SOCKS server. SOCKS64 extends existing SOCKS5 to allow transport layer translation to be achieved. But this tool dictates a SOCKS library installed as a prerequisite since it is based on SOCKS5. In SOCKS64, the hosts with SOCKS library installed forward the packet in one protocol to a SOCKS server. The server translates the flow into the

outgoing protocol to the other side and vice versa. As it works at application layer, it is free of MTU and fragmentation problem. However, in SOCKS64 every packet needs to be configured to go through the dedicated gateway where performs translation. Further, in SOCKS64, functions that request an IP address as one of the return values (e.g., `getpeername()` and `getsockname()` etc.) cannot provide the correct IP address as a return value due to address length difference between IPv4 and IPv6.

BIA (Bump In the API) is similar to BIS, but the translation in BIA is made between IPv4 APIs and IPv6 APIs. Besides, BIS is for node without IPv6 stack, but BIA is for dual stack node. BIA is implemented between the application and TCP/UDP layer of the stack on the host. When IPv4 applications on the dual stack communicate with other IPv6 hosts, the API translator detects the socket API functions from IPv4 applications and invokes the IPv6 socket API functions to communicate with the IPv6 hosts, and vice versa. Since it works at application layer, it supports IPsec and has no MTU fragmentation problem. But like other translation tools, BIA is still unable to work with IPv4 or IPv6 option. Further, BIA and BIS only perform application translation, while they do not mention any connectivity issue.

2.2.2 Evaluation of Transition Mechanisms

From the review of various transition mechanisms above, we realize that there is no one-fit-all mechanism during the transition period. However, to help assess them qualified for practical use or not, we apply following five main criteria: *robustness*, *compatibility*, *deployability*, *scalability* and *efficiency*.

Robustness

A robust protocol is still able to function correctly, with a certain degree of tolerance, even though factors that influence its operations are constantly changing. In the context

of complex system such as the Internet, factors such as bandwidth, routers, hosts, practically everything are subject to changes; they may inadvertently fail to operate properly, or assumptions made about them may not hold anymore, or they may even be malicious. Those possibilities introduce uncertainties on the Internet, and although it is impossible to eliminate all these uncertainties, we can still minimize them, or minimize the impact that may inflict upon a protocol.

In actual transition situation, the factor deciding a mechanism robust or not is whether the mechanism has single point failure or not. For example, in 6to4, if the border router is down, packets from IPv6 site will not be encapsulated, and obviously they are not able to traverse the IPv4 network.

Therefore, the mechanism which is free from single point failure can be seen as robust mechanism. Dual Stack mechanism provides IPv4 and IPv6 protocols to each node, BIS and BIA are self translator without the need of dedicate server, so they are all free of single point failure. Meanwhile, while SOCKS64 makes use of SOCKS gateway to perform translation, it may equip backup gateway. It thus avoids the single point failure. However, mechanisms based on the border router are inevitable of this problem such as all tunneling mechanisms, NAT-PT, IVI, and TRT.

Compatibility

Internet grows mostly in an evolutionary fashion, where transition periods, during which existing protocols are gradually replaced with newer protocols, are not uncommon. To encourage adoption, the new protocol must be compatible with the old protocol, or otherwise its connectivity is reduced to only small sets of networks that have implemented the new protocol. Compatible here means that the new protocol must be able to interoperate with existing protocols, and the interoperation must not negatively affect one party. Of course, the old protocol must also be compatible with the new protocol.

Generally speaking, the three types of transition mechanism are, to some extent, compatible with current IPv4 network. However, some of them are designed for special use, and they are thus not as compatible as other transition mechanisms. For example, the Teredo mechanism is designed to bring IPv6 connectivity to IPv6 nodes located behind IPv4 NATs. Therefore, it can operate only with specific NAT types, and its compatibility cannot be very well. Besides this special use transition mechanism, some other mechanisms lack compatibility due to their own limitation. For instance, we cannot run an IPv6 application on IPv4 native network though that node is Dual Stack node. Besides Dual Stack, all translation mechanisms are more or less lack compatibility as all of them have trouble in translating the application using the option field in IPv4 header or IPv6 header, which means some applications are not able to work with translation mechanism. In conclusion, we can see that all transition mechanisms except for Teredo, Dual Stack and all translation mechanisms equip good compatibility.

Deployability

Deployability means that a protocol can be easily integrated into the existing network infrastructure, without requiring fundamental or many modifications to existing protocols or network hardware. Thus, the main theme related to the deployability criterion is effortless integration. In this aspect, we present mechanisms that are useful to facilitate this effortless integration of a protocol into existing network infrastructure.

Actually, deployability is quite related to efficiency as low efficient mechanism cannot be deployed in practical use. Thus, mechanisms with low efficiency are not a deployable one. Dual Stack is a typical mechanism with low efficiency. Since both IPv4 and IPv6 are required to be equipped to support both protocols simultaneously in this mechanism, it has to keep two separate routing tables all the time, which means more memory and CPU power consuming. So Dual Stack mechanism is not suitable for largely deployment. Similarly, Tunnel Broker and Teredo also suffers this problem because this mechanism makes use of server for user to manually connect, register and activate tunnel to border router. Therefore, the overhead costs high due to the manual configuration.

Besides the efficiency, mechanism configuration also affects deployability. One deployable mechanism should be hidden from end user, which means end users are unlikely to notice the change after its deployment. Therefore, mechanisms have to modify existing network lead to a difficulty in deployment. And TRT mechanism is typical one of this type of transition technologies. In TRT, IPv6-only node needs special DNS server or modification of DNS resolver on initiating host to add certain prefix to IPv4-only node, after that they are able to communicate with each other. But this special DNS server and DNS resolver have to largely modify existing network component which is almost impossible to do. Similarly, BIS also equips this problem since it needs to modify kernel in order to insert its translator (three modules). In summary, it is hard to deploy the mechanisms which have to modify existing network such as TRT and BIS. At the same time, low efficiency mechanisms such as Dual Stack, Tunnel Broker, and Teredo are also unsuitable for practical deployment.

Scalability

The scalability of a protocol implies how well the protocol handles increasing load or complexity. Scalability is especially vital in Internet protocols because with the rapid growth of the Internet, Internet protocols' load also increases considerably.

From practical point of view, transition mechanisms with load balance are often more scalable than those without it. For example, NAT-PT is a mechanism without load balance which means every packet wanted to go to other network has to pass the NAT-PT gateway. As the number of devices initiating connections grows, so do the processing and site-maintenance loads, which can cause service degradation or failure. Like NAT-PT, Teredo and Tunnel Broker are also none load balance mechanisms which lack scalability. Oppositely, mechanisms with load balance such as TRT and SOCKS64 are with good scalability.

However, 6to4 technology is especially attractive on a large site basis though it has no

load balance. That's because in 6to4, it introduces no new IPv4 routing table entries and only one new entry into IPv6's global routing table. Unfortunately, 6to4 also suffers scalability issues which have been raised over for small site communication, especially node-to-node communication. ISATAP is just designed for solving this "node-to-node communication" problem in 6to4. However, ISATAP is just for node-to-node communication, it thus has no business with scalability issue. Similar to ISATAP, BIS, BIA and Dual Stack are all for single node, they therefore have no relation with scalability as well.

Efficiency

Efficiency concerns with how a protocol utilizes the resources that it has. An efficient protocol can obtain good performance, such as high throughput or small delay, without being excessively aggressive, but by making better use of available resources.

It is known that MTU and fragmentation problem will lead to a larger delay, so mechanisms have MTU and fragmentation problem inevitably suffer performance degradation. And as a result, mechanisms whose transition includes header translation all suffer performance degradation. At the same time, mechanisms work at TCP layer and application layer free from header translation equip better efficiency.

It is worth pointing out that Dual Stack mechanism also encounters performance degradation, though it has no header translation, due to high memory and CPU power consuming caused by keeping two separate routing tables simultaneously.

From the comparison we made above, we draw a table to simply assess the applicability of each mechanism from the five aspects.

	Dual Stack	Tunnel				Translation					
		6to4	ISATAP	Teredo	Tunnel Broker	NAT- PT	IVI	BIS	BIA	TRT	SOCKS6 4
Robustness	✓	✗	✗	✗	✗	✗	✗	✓	✓	✗	✓
Compatibility	✓	✓	✓	✗	✓	✗	✗	✗	✗	✗	✗
Deployability	✗	✓	✓	✗	✗	✓	✓	✗	✓	✗	✓
Scalability	-	✗	-	✗	✗	✗	✓	-	-	✓	✓
Efficiency	✗	✗	✗	✗	✗	✗	✗	✗	✓	✓	✓

Table 2-2 Comparison of Transition Mechanisms

In Table 2-2, the five characteristics of every transition mechanism are roughly depicted. Attention must be paid that in a good and complete protocol design, all the criteria must be considered. However, there are two issues that complicate the matter. First, all the criteria except efficiency are not exactly measurable. For example, no explicit metrics can be used to calculate the robustness or deployability of a protocol. It is up to the protocol designer's intuition and insights to determine whether a certain criterion is adequately considered. Second, the criteria are correlated with each other most of the time. It is impossible to design a protocol that performs extremely well in all criteria; instead, some tradeoffs are often necessary. A certain criterion may need to be sacrificed in order to provide better performance in another criterion. Thus, besides considering each of the criteria, the relationship between the criteria must be observed.

From the review and comparison, we find that with the significant work carried out under the IETF umbrella, the connectivity of IPv6 network is no longer a hurdle now. Instead, transition of application becomes critical obstacle in current transition phase [42]. Specifically, how to provide seamless transitions for a huge number of existing IPv4 applications and make all of them able to run over both IPv4 and IPv6 networks without any modifications are calling for great challenges and efforts.

Chapter 3

Veno2 Architecture Proposal

3.1 Idea Proposal

The conceptual Veno2 architecture, as illustrated in Figure 3.1, is a quite simple overlay network. It is composed of Veno2 *gateway* and Veno2 *agent*. Seeing the figure, Veno2 gateway is a Linux/Unix based appliance deployed in each organization's intranet, and Veno2 agent is client software installed on each of end users' hosts [54].

Figure 3.1 Veno2 Architecture

Inherently, Veno2 gateway is one kind of middleware server that can provide various session layer services, such as secure data forwarding, IPv4/IPv6 traffic relaying, auditing, and access control. Normally, there is one or more Veno2 gateways deployed in an organization's intranet and they are managed by the organization's administrator.

The responsibility of Veno2 agent is to enhance various session layer functionalities, i.e. communication security, IPv4/IPv6 transition, routing path selection of the applications (running on end users' hosts). In order to realize these functionalities, Veno2 agent supervises applications launched by end user and additionally performs extra network operations in session layer required by those session layer functionalities.

Essentially, Veno2 gateway and Veno2 agent cooperate together to form an overlay infrastructure over existing IP router based Internet [55]. In this overlay infrastructure, the nodes are composed of Veno2 agents and Veno2 gateways, and they are connected by virtual or logical links, each of which corresponds to a path, probably through many physical links. The virtual link not only represents the connectivity, but also embodies various session services, i.e. security and IPv4/IPv6 transition, implemented between two nodes. More specifically, Veno2 gateways are like proxy servers offering various session services, and Veno2 agents are clients that may or may not use those services.

Being different from clean-slate proposals, such as NGN [18] and NGI [19], our overlay infrastructure can be built up without modifying any existing IP infrastructure, such as IP routers and routing rules. Thus our proposal can dramatically reduce the barrier of deployment in practice. Deploying the infrastructure is a straight-forward process that typically takes less than a day and only involves installing Veno2 gateways in an organization's intranet and installing Veno2 agent on each end user's host.

Techniques like NAT or VPN requires its gateway server to be deployed at the edge of a network in order to capture and process all the traffic from or to the network. Such requirement limits the deployment choices that an organization's administrator can select. Being different, Veno2 gateway can be deployed at any place of the Internet and not necessarily required to deploy at the edge of an organization's intranet. Such open architecture allows additional integration with leading VPN-solutions and existing authentication servers, hence enabling users to leverage their existing network facilities and legacy network technologies.

Because Veno2 architecture works in session layer, it is compatible with various lower layer middle-box techniques, i.e. VPN, firewall and NAT. Such compatibility is critical for deployment in today's Internet, where VPN, firewall and NAT are already widely deployed. Moreover, operating in session layer also makes Veno2 architecture transparent to diversified lower layer techniques, such as TCP/UDP/SCTP/DCCP in transport layer, IPv4/IPv6 in network layer, and wired/wireless in link layer.

Most recently, overlay network has been proposed as a way to add new functionalities into the Internet revolutionarily [56]. As it requires no significant change in the existing IP infrastructure, overlay network quickly becomes popular as a new entrant to the Internet market, and has been used for many applications, such as content distribution [58][59][60], routing [55][61], QoS [62][63], multicast [64][65], security [66] and bridging multiple address spaces [67].

However, all these proposals are separately designed for specific applications, and thus each of them requires dedicated network resources (e.g. client agents or servers) to be deployed for each correspondent overlay network, which is usually redundant and sometimes even conflict. In addition, the general use of overlays is further hindered by tedious and complicated nature of porting legacy applications or equipping new applications with support for each particular overlay network.

Our Veno2 proposal provides a generic overlay infrastructure that can be used to support a variety of overlay services in a deployable fashion. In particular, as Veno2 infrastructure can integrate various overlay networks, not only the deployment cost is greatly reduced to one-time deploying for multiple overlays, but also the resources demands from different overlays can be centrally managed and contentions among them can be avoided. Moreover, Veno2 agent can be transparently deployed in end-user hosts without modifying both application software and operating system, which further facilitates deployment and improves user acceptance.

Several previous solutions, e.g. Planet-lab [68], Opus [69], and X-Bone [70] also aim to provide generic overlay services for many applications. However, these solutions are not transparent as they require modifying application software, and hence are not widely deployed or only used as testbed but not in real Internet. In contrast, OCALA [71] provides a transparent solution for convergence among different overlays without changing legacy applications, which is similar to our proposal. The difference is in that OCALA is designed to support the mutual communication among different existing overlays, while our proposal aims to augment functionalities into the existing Internet infrastructure in a deployable way. In other words, OCALA focuses on application services, while our proposal fundamentally targets at the Internet infrastructure. For instance, our proposal clearly specifies how to deploy our overlay components (Veno2 gateway and Veno2 agent) in different network topologies. Moreover, OCALA organizes overlay services horizontally and hence cannot apply multiple overlay services to one connection, while our proposal can achieve this through composing different overlays in a vertical stack in a similar way as protocol stack (as explained in details in Section 3.2.2), which is more flexible and configurable.

3.2 Design Principles in Veno2 Agent

Veno2 agent is software module installed on each of end users' hosts. It is responsible for enhancing various session layer functionalities, i.e. network security, IPv4/IPv6 transition, and routing path selection of the applications (running on end users' hosts). The design of Veno2 agent is centered on three main goals:

- ◆ **Transparency:** Neither application software nor operating system is required to be modified (recompiling or relinking) in order to support session layer functionalities of Veno2.
- ◆ **Extensibility and flexibility:** New session layer functionalities can be easily integrated into Veno2 agent, and several functionalities can be composed

together according to user-specified configuration.

- ◆ Application customization: Veno2 agent should be able to customize application protocol to satisfy the requirements of certain session layer functionality.

The following sections present how the design of Veno2 agent achieves these three goals.

3.2.1 Transparency through Code Interposition

Transparency means the ability to enhance functionalities of an application without any change (i.e. recompiling or relinking) or reconfiguration of either the correspondent client/server software or underlying operating system. Essentially, transparency here implies that applications are oblivious to the existence of underlying Veno2 agent, and the kernel of operating system needs no change.

This transparency feature of Veno2 agent is critical to the deployability of Veno2 architecture. Firstly, it ensures that existing application software needs no change. As a result, deploying Veno2 agent becomes a simple task involving just installing the Veno2 software in user's computer, and applications can simply be launched by just one click. Because a user is not required to modify or reconfigure the application software, the user's daily using habits are well kept. Second, transparency also allows non-administrative user to deploy Veno2 agent since the operating system is not modified. This significantly broadens the deploy environments of Veno2 agent. For instance, users can still use Veno2 agent through web downloading or thumb drive even in public environments, such as in shopping center, library or airport, where users typically do not have administrative privilege.

To realize such transparency, Veno2 agent employs code interposition technique, which allows intercepting and reimplementing library functions. By intercepting socket library functions called from application software, Veno2 agent is able to supervise network operations of the application and integrate additional session layer functionalities into the

application. For details about the code interposition technique, please refer to Section 4.2.

3.2.2 Extensibility and Flexibility by STREAMS Framework

There are many session layer services existing in the Internet, such as network security, IPv4/IPv6 transition, application layer routing, multi-homing, QoS and etc. And new session layer services are frequently proposed by researchers and engineers. Considering the great diversity and mutability of session layer services, Veno2 agent should be extensible to easily incorporate these services. In addition, it is not uncommon for different session layer services to be composed together to provide compound functionality. Consequently, Veno2 agent should also be flexible to allow easily composing services.

To provide a flexible and extensible system, the Veno2 agent borrows the basic concepts from System V STREAMS [73][74], extends and adopts them to be the Veno2 STREAMS Framework. The Veno2 STREAMS Framework provides a protocol composition framework, which allows users to dynamically extend an application's protocol stack in session layer by composing additional protocols into the stack. It shares commonalities with System V STREAMS. However, System V STREAMS focuses on transport and network protocols in kernel space, while Veno2 STREAMS focuses on application protocols in user space. For more details about STREAMS Framework, please refer to Section 4.3.

3.2.3 Application-dependent Customization

Some applications may require Veno2 agent to customize correspondent application protocols in order to achieve certain session layer functionality. For example, applications like email and instant message (i.e., MSN, ICQ and QQ) follow the client-server-client

communication model, where two clients communicate with each other through one or more intermediate servers. In order to protect these applications, Veno2 agent should be able to recognize the application protocols, and just encrypt the part of data that is only relevant to clients, leaving the data relevant to intermediate server to remain unencrypted so that the intermediate server can operate on such unencrypted data. Another example is to perform IPv4/IPv6 transition for applications with embedded IP address information, such as FTP, SNMP and SIP. Veno2 agent should be able to recognize application protocols and replace the embedded IPv4 address with correspondent IPv6 address, and vice versa.

To facilitate customizing application protocol, Veno2 agent provides a generic parser generator, with which developers can easily generate a parser for parsing specific application protocol. Besides, Veno2 agent allows callback functions to be registered to convert application data according to requirements of specific session layer functionality. For instance, security module in Veno2 agent may register callback functions for encrypting/decrypting application data, while IPv4/IPv6 transition module may register functions for replacing embedded IPv4 address with IPv6 address and vice versa. For more details about application-dependent customization, please refer to Section 4.4 for its implementation, and Section 5.1.4.1.1 and Section 5.2.3.1.1 for its application in security and IPv4/IPv6 transition.

3.3 Design Principles in Veno2 Gateway

Inherently, Veno2 gateway is one kind of middleware server that can provide various session layer services, such as secure data forwarding, IPv4/IPv6 traffic relaying, auditing and access control. Normally, there is one or more Veno2 gateways deployed in an organization's intranet and they are managed by the organization's administrator.

In the following sections, we discuss how Veno2 gateway is discovered by Veno2 agent

and achieves efficiency and reliability.

3.3.1 Gateway Discover

When a Veno2 agent installed in end host starts to run, it should be able to discover the information of its local Veno2 gateway deployed inside the correspondent organization's intranet. In addition, when a Veno2 agent attempts to access a remote host, it should be able to quickly discover the information of the remote Veno2 gateway correspondent to the remote host.

In order to enable quickly discovering gateway, we leverage the existing DNS [75] system by using the DNS prefixing technique [76], which obtains the name of the Veno2 gateway for a domain by adding a special DNS prefix (i.e. *venos*) to a domain name. For example, if the local domain for an end host is *ntu.edu.sg*, then the local Veno2 gateway for this host is *venos.ntu.edu.sg*, thus the Veno2 agent can find the local gateway address simply by resolving DNS name *venos.ntu.edu.sg*. Similarly, if the destination remote host is *www.google.com*, then the remote gateway address for this remote host is the correspondent address of DNS name *venos.google.com*.

With this DNS prefixing technique, the gateway discovery process becomes quite simple and fast, typically just involving one DNS query. Besides, deploying the gateway is also very simple, as the administrator of the correspondent organization is only required to add one address in their DNS zone.

3.3.2 Efficiency and reliability

In Veno2 architecture, we follow the “fat client thin server” design. That is, Veno2 agent is responsible to complex work, such as routing path selection, topology discovery,

security process, and IPv4/IPv6 transition (as discussed in more details in Section 5.1.3 and Section 5.2.2); while Veno2 gateway performs only simple work, like data forwarding or encapsulating/decapsulating data securely. This is because that in nowadays Internet, end hosts become more and more powerful to be capable of complex processing, while intermediate servers and gateways are typically bottleneck.

By moving complexity from Veno2 gateway to Veno2 agent, we make Veno2 gateway more efficient as it is only required to conduct simple process.

Reliability of Veno2 gateway is achieved by allowing organization administrators to easily deploy more Veno2 gateways to avoid single point failure. The Gateway Discover mechanism discussed in last section also assists to achieve automatic load-balance among multiple gateways through round robin DNS [77]. Moreover, Veno2 agent employs fallback mechanism, which automatically switches to use an alternative gateway if the primary gateway fails.

Chapter 4

Veno2 Implementation

4.1 Overview

The implementation of Veno2 architecture consists of two parts: Veno2 *gateway* software, which is a server application running in a Linux/Unix machine, and Veno2 *agent* software, which is client software installed by users in their own machines.

In the Veno2 architecture, we design Veno2 gateway to be simple by merely providing forwarding functions. So the implementation of Veno2 gateway is quite straightforward, and is similar to the implementation of existing proxy servers (i.e. SOCKS server or Squid server). Specifically, we implement Veno2 gateway using C language on Linux/Unix platform.

Figure 4.1 Veno2 agent components

Compared with Veno2 gateway, the implementation of Veno2 agent is much more complicated. As shown in Figure 4.1, Veno2 agent contains three components: VS-Stub, VS-Loader, and VS-Agent.

VS-Stub is a shared library being loaded into the target application process. It intercepts network function calls (socket APIs) and converts them to special requests, which are then submitted to VS-Agent. VS-Agent is a daemon program. It waits for requests from VS-Stub, processes them and returns processing results to VS-Stub. Moreover, VS-Loader provides GUI interface to allow users to load VS-Stub into target application process and control behavior of VS-Agent.

For example, when user starts VS-Loader program, VS-Loader automatically creates VS-Agent (i.e. vs-agent.exe) daemon process in background. Later, when the user chooses to run an application program (i.e. Internet Explorer) from VS-Loader, VS-Loader creates the application process and automatically loads VS-Stub (i.e. vs-stub.dll) into it. As a result, when the application process invokes a socket operation, VS-Stub intercepts it, converts it to a message, and submits it to VS-Agent. VS-Agent then conducts real socket operation and returns result to VS-Stub.

Figure 4.2 Key techniques in VenoSec agent.

Three key techniques are employed for implementing the Veno2 architecture. As shown in Figure 4.2, VS-Stub and VS-Loader leverages API (Application Program Interface) interception technique (examined in Section 4.2) on both Linux/Unix and Windows platforms. Moreover, VS-Agent follows the STREAMS Framework (discussed in Section 4.3), which makes Veno2 agent extensible and flexible. At last, both VS-Loader and

VS-Agent can utilize Application Transformer to facilitate transforming application data according to specific application protocol as well as requirements of particular session-layer services: such as encryption for security, or transiting IPv4/IPv6 address.

4.2 API Interception

API (Application Program Interface) interception [78] is a technique that can trap a function and modify its behavior. With this technique, one can extend the functionality of a binary program, without access to the relevant source code. Veno2 agent relies on API interception technique to achieve deployability. By intercepting socket functions invoked from application, Veno2 agent can transparently enhance the application's functionality (i.e., security and IPv4/IPv6 transition). In fact, users can easily deploy Veno2 agent in their own machines by simply installing the Veno2 software, even if the users don't have administrator privilege.

In the following sections, we describe the API interception technique in Windows and Linux/Unix platforms and how we apply it in the Veno2 agent.

4.2.1 Windows API Interception

There are many methods to hook API on windows platform [79]. Some act at the very bases of the operating system by sitting in kernel land, under the privileged ring 0 mode. Some others run in user land, under lower privileges in ring 3, which target directly the user's applications instead of the system itself.

Since the user land methods are more portable and deployable, we plan to employ this kind of methods to build the hook module in our system. Actually, the hook module is designed to be able to run under lowest privileges for a given account under windows. It

doesn't even use any administrative privilege to be able to perform its hooking as it resides directly inside processes that are owned by the current user.

The hook procedure is done in two steps. First, we inject our codes into the target application process owned by the current user (called code injection). Second, we replace the interested functions by our-supplied ones (called code interception). These tricks are performed at run time against a running process rather than on hard disk binaries since it allows to work around the windows file protection, antiviral and checksum tools as well.

In the following, we introduce different mechanisms to realize each hook step.

Code Injection

Altering the behavior of a process requires breaking into its memory space in order to execute some code to do the job. Unfortunately, Windows performs checks to prevent an application to read or write memory of another application without its permission. Nevertheless the Windows programmers included several ways to bypass the native inter-process protection so patching other processes' memory at runtime is a true possibility. Currently, there are three popular mechanisms to inject code into another process's address space: Windows Hooks, *CreateRemoteThread*, and Manipulating Thread's Context [80].

Code Interception

Injecting code into the address space of an external process is a key element of API hooking technique. It provides an excellent opportunity to have control over target process's activities. However it is not sufficient to have the code injected if we want to hook API function calls within the process. Moreover, we need to intercept interested API functions and replace them with our own. This is provided by code interception mechanisms.

Code interception aims to redirect another program's function when it is loaded in memory. For the target program, everything takes place as if it had called the desired functions as usual. But in fact the call is redirected to the replacement API. As for code injection techniques, there are three popular code interception techniques: Redirecting IAT [82][83][84], Redirecting EAT, and Inserting JMP [85].

4.2.2 Linux/Unix API Interception

Library Preload

On Linux/Unix platforms, users can easily intercept API by setting a LD_PRELOAD environment variable [86]. LD_PRELOAD enables users to specify additional shared libraries that are loaded when executing dynamically linked programs. Typically these preloaded libraries will override or intercept functions defined by other (regular) shared libraries. This allows users to modify the behavior of existing programs and libraries, without requiring accessing to the relevant source code.

For example, if user wants to intercept *connect()* function, he only needs to create a shared library, say *connect.so*, which implements and exports a *connect()* function, and set the LD_PRELOAD variable to equal to the shared library (*connect.so*). As a result, when users execute a program that invokes the *connect()* function, user modified *connect()* function, instead of the original one, will be executed first.

However, library preload technique can only effect applications that start with LD_PRELOAD being set properly. It cannot inject shared library into a running program. Also on some platforms, users are required to have root privilege to be able to use LD_PRELOAD technique.

Code Injection and Interception

Similar to interception on Windows platform, there are code injection and interception techniques on Linux/Unix platform.

The purpose of code injection on Linux/Unix is to inject a shared library (.so file) into a target process. This is usually realized through the `ptrace` system call, which can suspend a running process and change its code image in memory [87]. Specifically, we first suspend the target process by attaching to it (through `PTRACE_ATTACH`). Then we allocate a part of memory in the target process, which contains codes to load the shared library, and we generate a `JMP` instruction that jumps to the beginning of this allocated memory. At last, we resume the execution of target process by detaching from the target process. As a result, the target process will jump to the allocated codes and load the shared library.

On Linux/Unix, code interception can be realized through overwriting GOT (Global Offset Table). This technique is similar to the Redirecting IAT technique on Windows (see last section). For each external function used by a program or library, there is a correspondent entry in the GOT (of the program/library) that specifies the actual address of the external function. Consequently, to intercept a function, we only need to find its correspondent entry in GOT and modify the content of this entry to be our replacement function.

Another code interception technique is similar to the Detours technique on Windows (see last section). That is, a `JMP` instruction is inserted to the beginning of target function, which jumps to replacement function.

4.2.3 Applying API Interception in Veno2 agent

In Venos2 agent, VS-Loader employs code injection technique to load shared library into a target process, and VS-Stub makes use of code interception technique to intercept socket API functions and redirect them to VS-Agent.

On Windows, VS-Loader implements the *CreateRemoteThread* mechanism for code injection, and VS-Stub utilizes the *Inserting JMP* mechanism from the Detours technique. On Linux/Unix, LD_PRELOAD is used for both code injection and code interception.

Either *Inserting JMP* on Windows or *LD_PRELOAD* on Linux/Unix causes all socket function calls to be redirected to replacement functions. However, sometimes, we expect socket function calls from certain third-party library not to be redirected. In this scenario, VS-Stub can make use of *Redirection IAT/GOT* mechanism to suppress intercepting socket APIs in certain library.

Consider the following situations shown in Figure 4.3. The system has already used Inserting JMP or LD_PRELOAD technique to intercept *connect()* API with *my_connect()* function. The *my_connect()* function further invokes *SSL_connect()* function exported from third-party library *libeasy32.dll*, and *SSL_connect()* in turn needs to call the original *connect()* API [45]. However, as shown in the following figure, when *SSL_connect()* calls *connect()*, the execution is actually redirected to *my_connect()* again, resulting in invoking loop. In fact, we expect *SSL_connect()* to call the original *connect* function (*orig_connect()*).

Figure 4.3 Invoking Loop Problem

Our solution to this problem is to additionally apply *Redirecting IAT/GOT* mechanism to the *libeay32.dll*. The idea is that in the IAT/GOT section of *libeay32.dll* in the memory, we replace the address of imported socket function (i.e. *connect*) with the address of corresponding original function (i.e. *orig_connect()*). In this way, when any function in *libeay32.dll* invokes socket API, the execution will be correctly directed to the original API.

4.3 STREAMS Framework

The STREAMS system in Unix System V [74] provides a modular architecture for implementing network protocol stack between kernel or user space processes and device driver. One important feature of STREAMS is that it enables users to compose several network processing modules within the kernel to form a customized network stack. In such a modular way, users can modify and extend the functionality of network stack without kernel programming, compiling and link editing.

To provide a flexible and extensible system, the Veno2 agent borrows the basic concepts from System V STREAMS, extends and adopts them to be the Veno2 STREAMS Framework. The Veno2 STREAMS Framework provides a protocol composition framework, which allows users to dynamically extend an application's protocol stack by composing additional protocols into the stack. It shares commonalities with protocol composition systems, such as System V STREAMS, x-kernel [88], appia [89], cactus [90]. However, Veno2 STREAMS focuses more on application protocols in user space and the integration with VS-Stub.

The following sections describe the basic concept and architecture of Veno2 STREAMS Framework.

4.3.1 Architecture

As illustrated in Figure 4.4, Veno2 STREAMS Framework consists of five components: Modules, Channels, Connection Control Blocks (CCB), Messages, and Squeues.

Figure 4.4 Architecture of STREAMS Framework

STREAMS Module

A STREAMS *module* is a processing unit which does “black-box” processing on data that passes through it. Usually, one module represents a specific protocol and several modules chain together to form a protocol stack. For example, a SSL module can implement SSL protocol by conducting authentication and data encryption, or a SOCKS module can implement SOCKS protocol by relaying data through SOCKS server.

A module consists of both static and dynamic data for implementing a protocol. Static data may contain processing routines, i.e. encryption routines, static variables, and username and password for authentication. Dynamic data may contain mutable information for the whole module, such as number of connections processed by this module, or routing table for a module that conducts routing function. Note that data

relevant to specific connection is maintained in the Connection Control Block (CCB), discussed later in this section.

Modules interact with each other through exchanging messages between them. Each module has two message processing functions: *wput()* processes downstream messages from upper module, and *rput()* processes upstream messages from lower module. For example, for an encrypt/decrypt module, its *wput()* function is called to encrypt the data sent by application, and its *rput()* function is called to decrypt the data received by network card.

There are two special modules: head module and driver module. Head module locates on the top of STREAMS Framework, as shown in Figure 4.4. It specifies the interface between VS-Stub (the stub library loaded in target process) and VS-Agent (the daemon process conducting actual network operations). Specifically, it is responsible for translating socket requests received from VS-Stub into messages to be passed down to lower module, and for translating upstream messages received from lower module into socket responses to be returned to VS-Stub.

Driver module is the bottommost module in the STREAMS Framework, and specifies the interface between VS-Agent and socket layer of the operating system. It translates downstream messages received from upper module into proper network operations, and translates network data and events collected from lower socket layer into upstream messages to be passed to upper module.

In Figure 4.4, there are four modules: head module, module 1, module 2, and driver module.

STREAMS Channel

A STREAMS *channel* represents a protocol stack, consisting of several linear connected protocols. The channel specifies which protocols compose the protocol stack and the

order they chain together. Specifically, a channel is an ordered list of modules. The first and the last module must be head module and driver module respectively. Besides, there may be zero or more intermediate modules, each of which represents a specific protocol.

For example, a channel that only uses SSL protocol should be [head_module, SSL_module, driver_module]. In this case, data sent by application will be encrypted before transmitted into network, and received data will be decrypted before delivered to application. A more complex channel is [head_module, SSL_module, SOCKS_module, driver_module]. Through this channel, data sent by application is first encrypted then relayed to target through SOCKS server, and data returned from target is first received reversely through SOCKS server and then decrypted before delivered to application.

Channels enable users to configure distinct protocol stack for each network connection. Here, the term – “network connection” means a communication endpoint correspondent to a socket identifier (either TCP or UDP). A network connection is bond to a specific channel according to user-specified rules. Currently, the only supported rule is the name of application. For instance, users can configure all ICQ connections to go through a SOCKS server, or configure all connections from Firefox browser to be protected with SSL. As illustrated in Figure 4.4, Channel A consists of all 4 modules, while Channel B is composed of head module, module 2 and driver module.

Connection Control Block (CCB)

Connection Control Block (CCB) represents a network connection within a specific module. It contains private state information for the module to process the network connection, such as the target address, or the session key for encryption. A module may contain a set of CCBs, and each CCB is for a particular network connection. Also, a network connection usually has multiple CCBs, and each CCB is correspondent to one module. As shown in Figure 4.4, connection 1 is bond to Channel A, thus has 4 CCBs, each for one module. Connections 2 and 3 are bond to Channel B, and have their relevant CCBs in head module, module 2 and driver module.

STREAMS Message

STREAMS messages are the communication medium among modules in the STREAMS Framework. They convey control information or network data exchanged between VS-Stub and VS-Stub and between modules within VS-Agent.

Figure 4.5 STREAMS message structure

Figure 4.5(a) shows the structure of a STREAMS message. A message consists of three components: a message header (*VsMsg*), a data block header (*VsDataBlock*) and a data buffer. The message header contains a pointer (*data_block*) to the associated data block, and specifies the current read and write pointers (*rptr* and *wptr*) within the data block. It also contains other information relevant to the message, such as message type, associated channel id and connection id. The data block header describes a data buffer, by specifying the beginning and end of the buffer. The data buffer is a block of continuous memory containing actual data.

The message structure shown in Figure 4.5(a) helps efficiently utilizing memory and improves performance of STREAMS Framework. First, messages are exchanged between modules by reference to reduce memory-to-memory copy. Besides, data block employs reference count technique, which maintains a reference count identifying number of associated messages, and free the data block and its data buffer when reference count reaches zero. By referencing to the same data block, multiple messages can share a

common data buffer, thus save memory usage by avoid unnecessary memory copy (as illustrated in Figure 4.5(a)). Moreover, several messages can chain together (through a *cont* pointer) to form a composed message, so that no need to allocate a big buffer to hold all the data (see Figure 4.5(b)).

Type	Description
MSG_TYPE_DATA	I/O data message.
MSG_TYPE_CTRL	Socket control or event information exchanged between VS-Stub and VS-Agent or between modules within VS-Agent.
MSG_TYPE_INTL	Control or event information exchanged internally between modules within VS-Agent.

Table 4-1 Message types

There are three types of message (as shown in Table 4-1): data message (*MSG_TYPE_DATA*), control message (*MSG_TYPE_DATA*), and internal message (*MSG_TYPE_DATA*). Data message contains I/O data that applications send or receive. Control message encapsulates control information exchanged between VS-Stub and VS-Agent and between modules within VS-Agent. Internal message contains control information internally used by VS-Agent.

STREAMS Squeue

STREAMS squeue borrows the ides from Solaris squeue [91], which provides a simple and efficient mechanism for thread management. Here, threads management means how to associate threads with various STREAMS entities, such as modules, connection control blocks, and messages. There are two classical mechanisms for threads management: *vertical* threads model and *horizontal* threads model.

Horizontal threads model associates each module with a separated thread. This horizontal model is consistent to the layered architecture of STREAMS Framework, thus simplifies the design and implementation. Also it avoids synchronization complexity for multiple

connections to access a module's data, since only one thread is allowed to access the module's internal data. However, as number of modules is relatively small, the horizontal model only provides limited parallelism. Besides, each message is served by a new thread when traversing from one module to another, which incurs great context switch overhead.

In *vertical* threads model, a separated thread is associated with each incoming and outgoing message. It can achieve high parallelism as each message is processed simultaneously, and can avoid context switch overhead when the message traverses through different modules. However, performance may suffer when there are a large number of messages generated within a short period, which causes too many threads to be created. A variation of vertical threads model is to associate a separated thread with each connection. But it also has the same too-many-threads problem when there are a lot of connections.

STREAMS queue combines horizontal and vertical mechanisms in a way that reserves their advantages but avoid the disadvantages. Queue guarantees that only a single thread can process a given connection at any given time, thus it serializes the access to Connection Control Blocks (CCBs) by multiple threads. Consequently all data structures used to process a given connection can be accessed without additional locking, which reduces context switch overhead. Besides, the number of queues depends on the number of CPUs available in a computer, thus it will not induce the problem of generating too many threads as vertical threads model does.

Specifically, each queue maintains a queue of tasks and the identifier of the thread that is currently processing tasks in the queue. When a thread places a task into the queue, it will process the task by itself if no other thread is processing the queue. Otherwise, it queues the task at the tail of the queue and continues to do other task, and the thread currently processing the queue will process the task later.

In STREAMS Framework, each connection is associated with a correspondent queue. As shown in Figure 4.4, connection 1 and connection 3 are associated with Queue 1,

while connection 2 and connection 4 are associated with Squeue 2.

4.4 Application Transformer

Some applications may require Veno2 agent to customize correspondent application protocols in order to achieve certain session layer functionality. For example, to protect MSN chat communication, Veno2 agent should be able to recognize the MSN protocol, and only encrypt the chat content encapsulated in MSN messages. Another example is to perform IPv4/IPv6 transition for applications with embedded IP address information, such as FTP. Veno2 agent should replace the embedded IPv4 address with correspondent IPv6 address, and vice versa.

To facilitate customizing application protocol, we design and implement the Application Transformer, with which developers can easily generate a parser for parsing specific application protocol. Besides, the Application Transformer facilitates transforming application data based on the parsing result and requirements of particular session-layer services: such as encryption for security, and modifying IPv4/IPv6 address information for transition.

The following sections describe Application Transformer. Section 4.4.1 specifies the goals that the Application Transformer aims to achieve. Section 4.4.2 briefly presents the implementation. In Section 4.4.3, we take MSN as an example to show how to easily generate a parser.

4.4.1 Design Goals

Currently, developers parse application-layer protocols mainly by manually writing low level codes from scratch in an ad-hoc fashion. However, this usually involves tedious,

error prone and time-consuming works, such as writing code for matching protocol token with regular expression, and dealing with the complexity of inherent protocol grammars.

Tools like Lex [92] and Yacc [93] facilitate generating parser automatically through high-level meta-grammar language, which greatly simplifies developers' work. However, these tools are invented for parsing program language, and are not suitable for parsing network protocols for two fundamental reasons. First, these tools assume the data to be parsed is completely available when the parsers begin to parse. This assumption is valid for parsing program source code, but is invalid when parsing multiple, incomplete application data from network. In the extreme case, the data in a TCP stream may even reach byte by byte. Second, the common protocol syntax – a field in packet header to indicate the length of subsequent packet payload – cannot be expressed in most of those tools for parsing program language.

To tackle these problems, application parser generators, such as GAPA [94] and binpac [95] are proposed dedicatedly for parsing application protocols. However, these tools are proposed for just parsing and analyzing application data, but are not suitable for modifying the application data according to requirements from developers.

In order to facilitate transforming application in Veno2 architecture, we design Application Transformer with the following goals in mind:

1. Support partial match. Applications usually send and receive network data in multiple, incomplete data packets. In the extreme case, the data in a TCP stream may even reach byte by byte. Application Transform should be able to partially scan and parse incomplete input and resume where the parsing left off when more data reaches.
2. Support matching length field. It should be able to support the common protocol syntax – a field in packet header to indicate the length of subsequent packet payload.
3. Support modifying data. It should enable third-party developers to register

callback functions to modify application data according to their requirements.

4. Be flexible. It should support various application protocols and allow modifying any part of the application data.
5. Ease to use. It should be easy for third-party developers to generate parser and modify application on their demand.

4.4.2 Implementation

Application Transformer is implemented with a tree like architecture. Figure 4.6 shows a possible instance of the Application Transformer. Observing the figure, the transformer consists of three types of rules: Terminal Rule (TR), Non-terminal Rule (NTR) and Decorate Rule (DR). Each of these rules may match and operate over a subordinate part of the application data, such as a protocol token, an email address, a URL, or the whole message header or payload. These rules are organized together in a tree like architecture, which is correspondent to the grammar tree of the application protocol. In the following, we describe the basic components in the architecture: Terminal Rule, Non-terminal Rule, Decorate Rule, Action and Parser.

Figure 4.6 Architecture of Application Transformer

Terminal Rule

Terminal Rule matches and operates over two types of application data: data that can be matched by a specific regular expression, or data with a specific length. They are logically the smallest units to form the grammar of application protocol and are shown as leaves in the parsing tree in Figure 4.6. It is worthwhile to note that terminal rule is different from the concept of terminal symbol. The latter usually refers to syntactic token in a grammar, e.g. variable or function names in program language. In contrast, terminal rule is used to group a set of data according to logical semantic, usually containing multiple tokens. In practice, developers should decide how to divide an application protocol into a set of terminal rules.

For example, developers can divide the MSN message shown in Figure 4.11 into terminal rules in two ways. First, the message can be divided into a set of lines, and each line is represented and matched by a correspondent terminal rule. Alternatively, the message can also be divided into message header and message payload, separated by two consecutive CRLF sequences. The actual dividing mechanism is based on developers' requirements.

```
MSG 764 N 168\r\n
MIME-Version: 1.0\r\n
Content-Type: text/plain; charset=UTF-8\r\n
X-MMS-IM-Format: FN=MS%20Shell%20Dlg; EF=; CO=0; CS=86; PF=0\r\n
\r\n
Chat content line 1.\r\n
Chat content line 2.\r\n
```

Figure 4.7 An example MSN message

Specifically, we employ the PCRE library [96] for implementing the underlying regular expression. Besides, when applying terminal rule to match a specific length of data, developers can dynamically set the value of the length at run time. This feature allows expressing the syntax – a field in packet header indicates the length of subsequent packet payload. We will show an example about this later in Section 4.4.3.

Non-terminal Rule

Non-terminal rule consists of multiple sub-rules, which are grouped together according to specific logical operator. Each sub-rule can be any type of terminal, non-terminal and decorate rules. Currently, we define and implement three logical operators: Sequential Operator (NT_SEQ) which concatenates the data matched by two sub-rules, Alternative Operator (NT_OR) which matches the data if it matched by any of the sub-rules, and Repetitive Operator (NT_REP) which matches a sub-rule for specific times.

For example, suppose the MSN message in Figure 4.7 is divided into message header and message payload, and is matched by terminal rules TR_header and TR_payload respectively. Then we can construct the non-terminal rule NTR_MSN to match the whole MSN message with: $NTR_MSG = NTR (TR_header, NT_SEQ, TR_payload)$.

Decorate Rule

Decorate rule is used to decorate a terminal or non-terminal rule for adding certain pre or post processing to the parsing result by the correspondent rule. We borrow the concept from the Decorate mechanism in design pattern. It enables developers to conveniently extend a rule's operation.

One possible usage for decorate rule is to separate the data input and buffer mechanism from the parsing process. For example, suppose we need to parse MSN message coming from two different data sources: network and local file. There is no need to build two parsing rules from scratch for each of data source. Instead, we can reuse the NTR_MSN rule created before and just build two decorate rules, DR_net and DR_file, for handling input from data source. Then, we can construct rules for the two data sources by simply decorating the NTR_MSN with correspondent decorate rules through: $DR_MSN_net = DR_net (NTR_MSN)$ and $DR_MSN_file = DR_file (NTR_MSN)$. Besides, the DR_net and DR_file can also be reused for any other rules that have network and local file as data

source respectively.

Action

An action accompanies a rule and specifies the callback functions to be invoked each time an instance of that rule is recognized. Developers can register actions over any of terminal, non-terminal, and decorate rules.

In Veno2 architecture, actions are widely used for recognizing application protocols and modifying application data for realizing session layer services. For instance, Veno2 agent needs to know the application associated with a data stream through actions registered with the rule for the correspondent application protocol. In addition, actions are also responsible for customizing application data in order to realize session layer services, such as encrypt/decrypt data, and replace IPv4 with IPv6 information and vice versa.

Another usage for action is to modify the behavior of the generated parser at run time. For example, action is required for parsing length field in packet header, which indicates the length of subsequent packet payload. When used in this scenario, the action is registered with the sub-rule for parsing packet header, and is responsible for obtaining the actual value of the payload length and setting the sub-rule for parsing payload to use such length value. Section 4.4.3 will show an example on this.

Parser

Parser is just a simple wrapper over the rule object we mentioned before. Specifically, it is implemented as shown in the following code. The parse function *vs_parse()* just indirectly invokes the parse function of the underlying rule *vs_rule_parse()*.

```
inline gboolean
vs_parse (VsRule *rule,      /* The rule for parsing. */
          gchar *str,         /* The base address of the data to be parsed. */
```

```

    gssize str_len,          /* The length of the data to be parsed. */
    gint str_start,         /* The offset of the data from the base address. */
    gpointer user_data,     /* User supplied data for extension. */
    GError **error)        /* Error code returned if fails. */
{
    return (vs_rule_parse (rule, &str, &str_len, &str_start, user_data, error));
}

```

Currently, we define a set of API functions for developers to use the Application Transformer, instead of defining a new meta-language for writing script file used for generating parser as many other parser generators do. This is because of two considerations: 1) we assume that the developers who use this Application Parser in Venos architecture are more or less familiar with C programming; 2) the API provided by the Application Parser is itself quite simple. As a result, we believe it is more convenient and flexible for developers to directly use the API with C programming language or any other program language bindings, and there is no need to create a new meta-language, which burdens developers with extra effort for studying the new language and compiling the meta-language script into actual code.

4.4.3 Example

In this section, we take an example to show how to use Application Transformer to realize security services for MSN communication.

```

01 /* Regular expression for parsing MSN header. */
02 #define MSN_RE_HEADER \
03 "MSG.*(<msg_len>{0,10})\r\n" \
04 "MIME-Version: 1.0\r\n" \
05 "Content-Type: *\r\n" \
06 "(?:{0,1}\r\n\r\n)*\r\n\r\n"
07
08 /* Create the rule for parsing MSN message. */
09 VsRule *MSN_rule =
10     NTR(TR(MSN_RE_HEADER, action_header),
11         NT_SEQ,
12         TR(TR_LEN, NULL),
13         action_whole_message);
14
15 /* Action callback when matching MSN header. */
16 gboolean
17 action_header (VsRule *rule,
18               gchar **strp,
19               gssize *str_lenp,
20               gint *str_startp,
21               gpointer user_data,
22               GError **error)
23 {
24     gint payload_len;
25     VsRule *TR_payload;
26
27     /* Get the length of MSN payload from MSN header. */
28     payload_len =
29         atoi (TR_fetch_named_field (rule, "payload_len"));
30
31     TR_payload = TR_get_next (rule);
32
33     /* Set the length for the rule parsing MSN payload. */
34     TR_set_length (TR_payload, payload_len);
35     return (TURE);
36 }
37
38 /* Action callback when a whole MSN message is matched. */
39 gboolean
40 action_whole_message (VsRule *rule,
41                       gchar **strp,
42                       gssize *str_lenp,
43                       gint *str_startp,
44                       gpointer user_data,
45                       GError **error)
46 {
47     gchar *str;
48
49     /* Encrypt/decrypt the MSN payload, and deliver it to
50      next module in veno2 architecture. */
51     str = do_crypt_data (*strp, *str_lenp, *str_startp);
52     putnext_data (str);
53     return (TRUE);
54 }
55
56 /* Function for processing all MSN data. */
57 gboolean
58 secure_msn_data (gchar *str,
59                  gchar *str,
60                  gssize str_len,
61                  gint str_start)
62 {
63     return (vs_parse (MSN_rule, str, str_len,
64                       str_start, NULL, NULL));
65 }

```

Figure 4.8 Example code of Application Transformer for protecting MSN

The code is shown in Figure 4.8. From the code, we can see that the process to implement the parser is quite straightforward. From line 9 to line 13 is the code to create the rule to be used in Application Transformer. Line 10 and line 12 construct the sub-rule for parsing MSN header and MSN payload respectively. In line 10, the *NTR()* function finally creates the non-terminal rule *MSN_rule* by grouping the sub-rules for MSN header and MSN payload with sequential operator.

From the code, we can also see how to parse length field in MSN header, which indicates the length of subsequent MSN payload. When the parser encounters a MSN header, the correspondent action *action_header()* will be invoked, which will obtain the value of the length field from the MSN header and set this value to be the length of the sub-rule for parsing MSN payload. As a result, the parser will continuously fill MSN payload until the specific length of bytes are read. Since, at this time, a whole MSN message is matched, the *action_whole_message()* function will be invoked, which encrypt/decrypt the MSN

payload data and deliver the result to next processing module in Veno2 architecture.

Chapter 5

Two Instances of Veno2 Architecture

This chapter contains experimental evaluation through two instances of Veno2 infrastructure: VenoSec (Section 5.1) and VenoSink (Section 5.2). For each of these two architectures, we first present their mechanisms when deployed in various environments; this is followed by technical framework describing the actual implementation; and finally, we show experiment results obtained from both testbed and real Internet.

5.1 VenoSec: Instance of Veno2 for Security Platform

5.1.1 Overview

There was little provision for security in the original design of the Internet [21]. As the Internet becomes more and more popular, a lot of cheating, virus, cheated online trading, and other dishonest behaviors are frequently spreading over the Internet. Network security has been drawing much more attention and concerns to network administrators and users.

Over the last few years, many security techniques have been developed to protect network communications. SSH [10], FTPS [43], HTTPS [16] and POPS [44] only protect

their correspondent applications; they cannot be applied to protect other applications. IPSec [13], SSL [14] and SOCKS [15] are proposed to provide generic security services for all applications. However, IPSec's inherent incompatibility with NAT (Network Address Translation) restricts its deployment [35]. SSL is only widely used for Web application due to its requirement of modifying application code [45]. Also, SOCKS only protects communication between client and SOCKS server, but not full end-to-end path. As a result, all these techniques are not widely deployed for protecting end-to-end security.

In this section, we propose *VenoSec*, a deployable secure architecture, which aims to provide various security services for end users with very little deployment cost. There are no changes required on either application code or operating system kernel. The only need is for end-users to just install the VenoSec software in their computers, even for users without administrative privilege. In the following, we present network attacks and security services model, and explain how VenoSec works, its implementation detail and the experiences from deployment in real environments.

5.1.2 Network Attacks and Security Services Model

According to [46], an attack refers to "an assault on system security that derives from an intelligent threat, i.e., an intelligent act that is a deliberate attempt (especially in the sense of a method or technique) to evade security services and violate the security policy of a system." As such, there are many attacks that can be launched against the security of a system interconnected to the Internet. There are two categories of attacks: passive and active. A passive attack "attempts to learn or make use of information from the system but does not affect system resources," whereas an active attack "attempts to alter system resources or affect their operation" [46]. In the following, we overview and briefly discuss some exemplary network attacks.

Eavesdropping

An attacker is able to interpret and extract the information that the transmitted data encodes. For example, if two parties communicate unencrypted, a passive eavesdropper is trivially able to extract all information that is encoded in the data.

Traffic analysis

An attacker is not able to interpret and extract the information that the transmitted data encodes. Instead, traffic analysis refers to the inference of information from the observation of external traffic characteristics. For example, a timing attack on the SSH protocol can use timing information to deduce information about passwords since, during interactive session, SSH transmits each keystroke as a message. [

Identity spoofing

An attacker successfully masquerades as another by falsifying data and thereby gaining an illegitimate advantage. For example, in email address spoofing, the sender information shown in e-mails (the "From" field) can be spoofed easily.

Man-in-the-middle

An attacker makes independent connections with the victims and relays messages between them, making them believe that they are talking directly to each other over a private connection, when in fact the entire conversation is controlled by the attacker. For example, the "Trust-on-first-use" authentication mechanism of SSH is vulnerable to such attack.

Password recovery

An attacker recovers passwords from data that has been transmitted through a communication system. For example, an eavesdropper can easily learn the password in a Telnet session.

Data modification

An attacker can modify the data in the packet without the knowledge of the sender or receiver. Even if you do not require confidentiality for all communications, you do not want any of your messages to be modified in transit. For example, if you are exchanging purchase requisitions, you do not want the items, amounts, or billing information to be modified.

Replay

An attacker copies a stream of messages between two parties and replays the stream to one or more of the parties. Unless mitigated, the computers subject to the attack process the stream as legitimate messages, resulting in a range of bad consequences, such as redundant orders of an item.

Denial of service

An attacker prevents an Internet site or service from functioning efficiently or at all, temporarily or indefinitely. Perpetrators of DoS attacks typically target sites or services hosted on high-profile web servers such as banks, credit card payment gateways, and even root name servers.

Having identified the relevant security attacks to a network, the network operator can apply various security services and mechanisms to confront these attacks. In the following, we briefly present five key security services.

Authentication

Authentication indicates a sender (Alice) knows that the receiver she intends to communicate is the actual one (Bob) and meanwhile, the receiver (Bob) is quite sure that the other side is indeed Alice, not anyone else. In other words, the sender and receiver can confirm each other's identity and the origin/destination of the information. Authentication service is typically provided through key distribution and identity management systems.

Confidentiality

Confidentiality means that information is not made available, or cannot be understood if available, by anyone for whom it was unintended. It is usually achieved through encryption and decryption algorithms.

Integrity

Integrity indicates a receiver wants to make sure that the data he receives is the true data without any alternation in transit or in storage. On the Internet, all bits, either 0 or 1, look alike, it's trivial to tamper with in-transit messages. For example, an attacker can simply intercept the message from the wire, copy out the parts he likes, add whatever data he wants, and generate a new message to receiver. Integrity is protected through hash

algorithms.

Non-repudiation

Non-repudiation indicates the sender of a message cannot subsequently deny having sent it. Consider the situation that Alice and Bob have signed a contract, but later Alice may wish to abrogate unilaterally. She might simply claim that she had never signed it with Bob. Non-repudiation prevents Alice from doing so, thus protects the rights of a recipient (Bob). Digital signature is employed to protect non-repudiation.

Availability

Availability refers to the availability of information resources. An information system that is not available when you need it is at least as bad as none at all. For example, if the primary trading database for a securities brokerage firm is inaccessible, millions of dollars could be lost with every passing minute. Availability can be provided by techniques like load balance, resource cache and redundancy.

Network Attacks	Security Services	Security Techniques
Eavesdropping	Confidentiality	Encryption/Decryption
Traffic analysis	Confidentiality	Encryption/Decryption
Identity spoofing	Authentication/Non-repudiation	Identity management
Man-in-the-middle	Authentication	Identity management
Password recovery	Authentication	Identity management
Data modification	Integrity	Hash function
Replay	Integrity	Hash function
Denial of service	Availability	Load balance/Redundancy

Table 5-1 Relationship between network attacks, services and techniques

In summary, Table 5-1 shows the relationship between network attacks, correspondent security services and the techniques to provide those services.

5.1.3 VenoSec Mechanism

Figure 5.1 illustrates how VenoSec protects the communication between a client Host C and server Host S. Suppose the client and the server are in two enterprise networks far away from each other, and there may be various intermediate nodes, such as firewall, NAT device, or VPN server, along the path between the client and server. The client node is running different client applications, i.e. Telnet, HTTP or FTP, attempting to access correspondent server application running in the server node. In this case, attackers locating between the client and server may eavesdrop or modify the data transmitted between them.

Figure 5.1 VenoSec mechanisms for client-server application

With VenoSec, it's quite easy to protect the data exchanged between the client and server for various applications. As shown in Figure 5.1, users in both client and server install VenoSec software and launch client/server applications from VenoSec user interface in just one click. The VenoSec software on both sides cooperates together to establish a secure channel between them, which may involve various security techniques, such as authenticating each other, negotiating security primitives and parameters, and protecting application data by encrypting.

Essentially, VenoSec acts as a security agent for a host. Whenever an application in the host attempts to communicate with a remote application, VenoSec will be activated to take over the communication process and enforce user-specified security policy to the communication. For example, when user starts a *telnet* program from VenoSec user

interface and connects to a server, the connecting process will be intercepted by VenoSec. VenoSec checks the security policy and decides that the connecting process should be authenticated through Kerberos protocol [36]. As a result, VenoSec not only establishes a TCP connection, but also conducts Kerberos authentication with the server.

Note that in above example, all that users need to do is just to install the VenoSec software, and run applications from VenoSec user interface with only one click. There is no need to download and install secure software correspondent for each application; users can continue to use the software they get used to. Besides, since VenoSec does not affects operating system's kernel, users without administrative privilege can also install and run VenoSec. For application programmers, there is no need to modify their applications' code or any relink or recompile work. The same VenoSec software can be used to protect different end-to-end applications, even for newly invented one. Moreover, since VenoSec works in application layer, it is compatible with various intermediate boxes, such as VPN, NAT, and firewall.

Client-to-client communication through intermediate server

In the above section, we only discuss how communication directly between client and server is protected. In fact, besides the direct client-server communication model, there is another widely used client-server-client communication model, where two clients communicate with each other through one or more intermediate servers. For example, two email clients exchange emails through one or more email relay servers, and instant message clients, such as MSN, ICQ, and QQ, exchange chat messages through one or more instant message servers. In this section, we present how VenoSec protects the communication of client-server-client model.

Figure 5.2 VenoSec mechanisms for client-server-client application

Figure 5.2 illustrates two MSN clients C1 and C2, who exchange users' chat message through a MSN server S. When plain text MSN messages traverse from C1 to C2, attackers along the path C1-S-C2 can easily eavesdrop to obtain the MSN messages and learn users' private chat content.

To protect MSN communication, VenoSec software is installed on clients C1 and C2, note that no need to install VenoSec in MSN server S. VenoSec software on both sides is responsible to encrypt/decrypt MSN messages between them. However, VenoSec should not encrypt the whole content in a MSN message. This is because a MSN message contains two parts: a MSN header that MSN server S needs to read and process, and a MSN payload that are dedicated for clients. If VenoSec encrypts the whole MSN message including MSN header, the MSN server is not able to recognize the encrypted MSN header, thus discards the message. In fact, VenoSec only needs to encrypt the MSN payload to protect users' privacy, while leaves the MSN header to be plain text so that MSN server can process it.

Essentially, VenoSec works as an Application Level Gateway (ALG) [115] for client-server-client application, which can recognize application specific commands and offer security controls over them. When a client sends messages to another remote client through an intermediate server, VenoSec in the sending client parses the messages to

glean the payload data, which is only meaningful to clients and will not be touched by server. VenoSec encrypts the payload data and re-write the encrypted payload back to the messages that is sent to server. The server may modify some information in the messages, but the encrypted payload is kept untouched and is relayed to the remote client. VenoSec in the remote client parses the received messages to separate the payload data, and then decrypts them.

Considering the nontrivial work to implement a dedicated ALG for each client-server-client application, VenoSec provides a generic Application Transformer to facilitate parsing different application layer protocols and enforce security controls to these applications.

In the following sections, we discuss how VenoSec can be easily and incrementally deployed in two typical scenarios: individual user scenario, where no central authority or agent is available, and group user scenario, where there is trusted authority and Veno2 *gateway* is deployed to assist secure data relaying.

5.1.3.1 Individual User Scenario

In this scenario, there is no centralized authority or agent to cooperate the communication between users; each user individually interacts with another user. For example, Alice attempts to download a file from Bob through Bob's FTP server. In this case, no third part, but only Alice and Bob are involved in the whole process of communication.

Figure 5.3 illustrates how VenoSec is deployed to protect communication between individual users, who may or may not be in a same enterprise network. As shown in the figure, Enterprise A Network and Enterprise B network are far away from each other. There is no secure service being provided either between the two networks or within each network. In this case, attackers inside an enterprise network can mount attacks on

intra-network communication, i.e. between users A1 and A2, or between B2 and B3, and besides, inter-network communication, i.e. between A1 and B2, is vulnerable to both inside and outside attackers.

Figure 5.3 Deploy in individual user scenario

For this scenario, the deployment of Venosec is quite straightforward: the users, who expect to communicate securely between them, just install Venosec in their computers. For example, after A1 and A2 install Venosec, they can securely communicate with each other. Likewise, Venosec is deployed in B2 and B3 hosts to protect the communication between them. At this time, the communications between A1 and B2 is also automatically protected because they already installed Venosec. In the similar way, secure services are provided between A1 and B3, A2 and B2, and A2 and B3. In a word, communication between any two hosts that have Venosec installed is protected.

Since there is no central authority or agent in this scenario, individual users cannot authenticate each other through PKI (Public Key Infrastructure) [116] or other centralized certification server. Thus, for this scenario, the default authenticate method that we recommend in Venosec is the widely used simple and cheap "Trust-on-first-use" mechanism commonly associated with SSH and HTTPS with self-signed certificates [135].

5.1.3.2 Group User Scenario

In this scenario, each user is a member of a group of users who are managed by a common network operations entity. The users usually share common network infrastructure, such as routers and gateways, and can authenticate each other through a centralized authority, i.e. PKI [116] or Kerberos [36] server. Examples of this scenario include student or staff users in a campus network, and employee users in an enterprise network.

Figure 5.4 Group user scenario without Veno2 gateway

Figure 5.4 shows an Enterprise Network A and several enterprise users that are outside the enterprise intranet. Network A comprises various servers for the enterprise, such as email server, Web server and database server, and the users outside expect to remotely access those servers in Network A. We assume PKI server is deployed in Network A to provide authentication service for enterprise users.

One approach to deploy VenoSec in Figure 5.4 is quite similar as the individual user scenario (see last section): the users that need secure communication install VenoSec software in relevant computers. So enterprise administrators install VenoSec in servers inside Network A, and the remote users also install VenoSec in their own computers. Then both the communication among the remote users and the communication between

remote users and intranet servers will be protected. The difference from the individual user scenario is that now users can be authenticated reliably through the PKI server, instead of using the “Trust-on-first-use” method that is vulnerable during the first use.

Alternatively, another approach to deploy VenoSec is via the assistance of Veno2 gateway. As shown in Figure 5.5, the Veno2 gateway (VG) is a Linux/Unix server, locating in Network A. Still, VenoSec software is installed in each of the remote users’ computers, who expect to securely communicate with various servers in Network A.

Figure 5.5 Group user scenario with Veno2 gateway

The Veno2 gateway is designed to protect the communication path between itself and VenoSec software installed in users’ computers. Observing Figure 5.5, the communication paths between gateway VG and each of remote users are protected. It cooperates with the VenoSec software to provide secure relaying services for client applications. For example, assuming remote user A1 attempts to access the intranet Web server, the VenoSec software in A1 intercepts and encrypts the sent application data and transmit the encrypted data securely to Veno2 gateway VG, which then decrypts the encrypted data and relays the original application data to the Web server. The function of Veno2 gateway here is much like a proxy’s role, i.e. SOCKS5

The above deploying strategy does not need to install VenoSec software in application servers, and thus will not interrupt the service providing from those servers. It is suitable for situations where service interruption is not tolerable. However, this deploying strategy

only protects the communication path between clients and Veno2 gateway, leaving the path between Veno2 gateway and application server vulnerable to attacks from internal intranet. To solve this problem, administrators can choose to incrementally install VenoSec in particular servers that need more strict security controls, and protect them from internal attackers.

Essentially, VenoSec can be employed to realize VPN (Virtual Private Network) in the group user scenario. With incrementally deploying of VenoSec agents and gateways, VenoSec can even replace the functionality of original VPN with enhanced flexibility and security guarantee even in enterprise's intranet.

5.1.4 Technical Design

As mentioned in above section, VenoSec platform consists of two components: a Veno2 *agent* that is installed in each end of a communication, and a Veno2 *gateway* that is deployed for an enterprise network to provide secure relay service to the hosts inside. The following sections present how these two components are implemented.

5.1.4.1 Agent

Veno2 *agent* is developed by inserting a thin layer in TCP/IP model. Figure 5.6 illustrates its location which is between the application layer and the socket layer.

Figure 5.6 Architecture of Veno2 agent for VenoSec

By this layer, all socket function calls by applications can be intercepted, processed and at last proceed to transport layer through the socket layer. To some extent, we can say *VenoSec* layer is kind of an extension of socket layer; the purpose is to provide more powerful session functions in the existing TCP/IP model.

Observing Figure 5.6, the *VenoSec* layer consists of three components: application transformer, policy manager and security provider interface. In the following sections, we describe each of these components in details.

5.1.4.1.1 Application Transformer

Applications like email and instant message (i.e., MSN, ICQ and QQ) follow the client-server-client communication model, where two clients communicate with each other through one or more intermediate servers. In order to protect these applications, *VenoSec* is deployed in two client hosts. As mentioned in Section 5.1.1, simply encrypting all the exchanged data between clients does not work, because the intermediate servers are not able to recognize the encrypted data and thus regard the data as invalid. Instead, *VenoSec* should just encrypt the part of data that is only relevant to clients, and leave the rest of data relevant to intermediate servers to be still plain text.

Essentially, VenoSec works as an Application Level Gateway (ALG) that can recognize application specific commands and offer security controls over them.

Considering the nontrivial work to implement a dedicated ALG for each client-server-client application, VenoSec provides a generic Application Transformer, with which an application specific ALG can be easily built and integrated into the Veno2 agent. As shown in Figure 5.6, Application Transformer consists of two components: Application Parser and Application Converter.

Application Parser provides an engine to generate parser for parsing specific application protocol. In other words, a corresponding parser can be automatically created by composing a script.

Based on the parsing result by Application Parser, Application Converter aims to facilitate converting application data in order to satisfy security requirements. Specifically, it provides APIs for inserting, removing and modifying application data in a way that makes the result data be still consistent with the application protocol.

Figure 5.7 illustrates how to utilize Application Transformer for protecting MSN communication. MSN client application first creates a MSN message containing user's chat content to be sent to the peer user. The message is passed to Application Parser and is separated into two parts: MSN header that is relevant to MSN server and MSN payload that is relevant to client. Then Security Provider Interface is invoked for encrypting the MSN payload data. The Application Converter then reassembles the encrypted MSN payload with the MSN header, which involves updating some fields in MSN header, i.e. updating the message length from 71 to 79 as shown in Figure 5.7. Finally, the newly composed MSN message is sent to the peer user. Since the MSN payload is encrypted with a key only known by the two end users, no third party is able to learn the private chat content between them.

Figure 5.7 Using Application Transformer to protect MSN communication

Similarly, in the receiver side, the Application Parser separates MSN payload from MSN header and delivers them to the Security Provider Interface. The Security Provider Interface invokes appropriate cryptography mechanism for decrypting the encrypted payload in order to obtain the original payload data that contains the chat content. Then the Application Converter reassembles the original payload data with the MSN header and passes the result MSN message to MSN application. Finally, the MSN application displays the sender's chat message to the user.

Sometimes it is required to insert new application data into a communication. For example, if an encrypted MSN payload is too long to be accommodated into a single MSN message, a new MSN message should be composed and appended to hold the extra data. Besides, if two MSN users decide to perform authentication or key exchange between them, extra MSN messages should be composed and exchanged. The Application Converter guarantees that those extra application data is valid and consistent with the application protocol.

5.1.4.1.2 Policy Manager

Policy Manager provides a convenient, consistent and granular method for users to manage security policies in VenoSec. Here, security policy is a configurable rule that specifies the security requirements for particular application traffic. For example, a simple security policy may dictate that any MSN session between user vs1@msn.com and vs2@msn.com should be encrypted with a shared key by 3DES [136] algorithm.

A security policy is composed of *Traffic Selector* and *Security Descriptor*. The *traffic selector* defines the set of traffic associated with the policy. For instance, a coarse-grained *traffic selector* may encompass all traffic between a pair of hosts, while a fine-grained one may only contain FTP traffic between the two hosts. The *security descriptor* specifies how to protect a particular traffic. For example, a *security descriptor* may require mutual authentication through Kerberos and data confidentiality through AES [137] encrypting algorithm.

Observing Figure 5.6, Policy Manager consists of two components: Policy Database and Policy Selector.

Policy Database

Policy Database contains a list of security policies, ordered from high priority to low priority. Each security policy specifies which data traffic should be protected and what security services should be offered. In other words, the Policy Database controls the granularity and the intensity of protection.

For example, Table 5-2 shows a Policy Database containing 4 security policies. The first policy defines that any MSN communication with peer user vs@msn.com should be authenticated through “Trust-on-first-use” method, and protected with 3DES [136] encryption and MD5 HMAC (keyed-Hash Message Authentication Code) [138]. The

second and third policies specify how to protect Telnet or all TCP traffic with remote host 155.69.101.100 respectively. At last, the fourth policy is the default policy that dictates the protection for all other data traffic.

Order	Security Policy	
	Traffic Selector	Security Descriptor
1	MSN with peer vs2@msn.com	Trust-on-first-use, twofish, MD5
2	Telnet to server 155.69.101.100	Pre-shared key, AES, SHA1
3	All TCP with host 155.69.101.100	PKI certificate, blowfish, MD5
4	Default	Trust-on-first-use, blowfish, SHA1

Table 5-2 Example Policy Database

Since security policies in Policy Database are sorted by priority, previous policy with high priority may cause a subsequent policy to never be used. For example, if the second and third policies in Table 5-2 exchange their order, the new third policy will become useless because it only handles Telnet traffic which is already covered by the second policy handling all TCP traffic. As a principle, the security policy covering smaller range of traffic should have higher priority over the policy with wider covering of traffic.

Essentially, the Policy Database is similar to the network firewall rules in operating system. They both contains an ordered list of rules, with each rule specifies how to handle a particular set of data traffic. However, firewall rule can define only to accept, reject or discard the data traffic, while the Policy Database can additionally dictate to protect the data with security techniques. Besides, the Policy Database has more granularities control over the data traffic being selected, such as application type, system user account or application user id.

In VenoSec, the Policy Database must be consulted during the processing of all data traffic, either incoming or outgoing. Specifically, when VenoSec attempts to deliver any application data, it first requests the Policy Manager to find the appropriate security policy to be enforced to the data. The Policy Manager searches the Policy Database, and

selects the first matching security policy. Then VenoSec knows how to protect the application data.

VenoSec provides simple user interface so that users can conveniently manipulate the Policy Database. For example, users can insert a new security policy into the database, remove an existing policy from the database, modify a particular policy, and adjust the sequence of policies. Besides, the Policy Database should be able to detect and resolve the potential conflicts among policies inside.

Policy Selector

Policy Selector selects an appropriate security policy from the Policy Database according to the characteristic of application traffic. It sequentially examines each security policy in the Policy Database, and selects the first one that matches the application traffic. As mentioned previously in this section, each security policy contains a traffic selector that defines the set of application traffic associated with the policy. The Policy Selector can determine whether a security policy matches certain application traffic by comparing the application traffic against the traffic selector in the policy.

For example, suppose the Policy Database is the same as the one shown in Table 5-2. And suppose that the application data to be handled belongs to a Web client application accessing the server 155.69.101.100. In this case, the Policy Selector orderly examines the policies in the database and selects the third policy as matched.

Policy Selector controls the granularity of protection enforced to data traffic through the traffic selector. The traffic selector defines various metrics to describe the characteristics of data traffic. For instance, the metrics may be network layer information, such as source or destination IP address; or transport layer information, such as port number, being TCP or UDP; or even application layer information, such as the application type (MSN, Telnet, or FTP), system user account (administrator or normal user), and application user identifier (MSN id vs@msn.com or ICQ id 82345671).

Considering that different users may have specific criteria to select data traffic, the Policy Selector provides APIs for third parties to extend the selector by defining their own metrics. For example, suppose an enterprise develops a new network application with a specific format of user id, and expects to protect this new application with VenoSec. In this case, a new metric can be defined in the Policy Selector, which is able to select the data traffic associated with a specific user id for the new application.

In order to make VenoSec more reliable, the Policy Selector employs a fallback algorithm. That is: if the first selected security policy fails to offer security services (e.g. the peer's certification is unknown or expired), the Policy Selector does not stop processing but selects the second matched policy in the Policy Database, and employs it for protection. For example, suppose the Policy Database is the same as the one in shown in Table 5-2, and suppose the application data to be handled belongs to a Telnet application accessing the server 155.69.101.100. The Policy Selector will select the second policy. Assume authentication though the second policy fails due to the pre-shared key being modified by server. Then the third policy is selected for protection. Assume the PKI certificate of the server is expired and authentication still fails. At last, the default policy is selected and employed for protection.

5.1.4.1.3 Security Provider Interface

Security Provider Interface encapsulates all the cryptography mechanisms, algorithms and protocols for providing various security services, such as authentication, confidentiality, integrity, and authorization. Besides, other components in VenoSec, i.e. Policy Manager and Application Transformer, can access the security services offered by Security Provider Interface in a generic fashion, independent of various underlying security mechanisms.

There are many high-level security packages, such as SSL, Kerberos, IPSec and SET, which combines many different cryptography algorithms and techniques to provide various security services. The Security Provider Interface defines a generic interface to access the security services offered by these security packages without dealing with the underlying security protocols of the packages. At the same time, the Security Provider Interface also allows other components of VenoSec to take advantage of the low-level cryptography algorithms and techniques. For example, we can enforce SSL protection to a TCP connection through the Security Provider Interface without knowledge of the details of the SSL protocol. And we can also encrypt a block of data with 3DES through the Security Provider Interface.

Third party users can extend the Security Provider Interface by integrating their own security packages, protocols, or algorithms into it. Specifically, the relevant functions in Security Provider Interface are mapped to the implementation specific to the package, protocol or algorithm. In this way, these new packages, protocols or algorithms can be used by other parts of VenoSec without changing the callers' interface.

As shown in Figure 5.6, the Security Provider Interface consists of two components: *Authentication Provider* and *Data Protection Provider*.

The Authentication Provider is responsible for negotiating security mechanisms, authenticating each other, and initializing a security context. When two end-users attempt to communicate, the Authentication Provider is first activated to negotiate security mechanisms for performing authentication and protecting subsequent data, i.e. authentication algorithm, encryption/decryption algorithm, and data integrity algorithm. If the negotiation succeeds, the Authentication Provider proceeds to authenticate each other. According to the application's requirement, the authentication may be client-only, server-only, or mutual. If the authentication succeeds, the Authentication Provider creates a security context and stores the negotiated security mechanisms into it.

After the Authentication Provider initializes an appropriate security context, the Data

Protection Provider will then protect confidentiality and integrity for the data subsequently transferred in conjunction with the security context. Specifically, the Data Protection Provider in the sender side performs data integrity processing and encrypts the data before sending it out, and the Data Protection Provider in the receiver side decrypts the data and validates the data integrity check. The cryptography algorithms used in above processing are specified by the security mechanisms included in the security context.

We use SSL protocol as an example to illustrate how a security protocol is mapped into Authentication Provider and Data Protection Provider. The Authentication Provider for SSL protocol maps to both SSL Handshake Protocol and Change Cipher Spec Protocol, which negotiates protocol version as well as cryptography algorithms, optionally authenticates each other, and employs public-key technique to generate shared secrets. The Data Protection Provider maps to SSL Record Protocol, which performs compression, appending MAC digest, and encryption or decryption processing on the application data.

5.1.4.2 Gateway

In VenoSec platform, Veno2 *gateway* is a server application running in a Linux/Unix machine, deployed within an organization for providing secure forwarding services for users inside. As discussed in Section 5.1.3.2, Veno2 gateway is used in group user scenario, where users outside an organization intranet expect to access the servers inside the organization. Examples of this scenario include student or staff users accessing their campus network from outside, and employee users outside accessing resources within enterprise intranet. In this scenario, there is usually a centralized authority, i.e. PKI (Public Key Infrastructure) or Kerberos server, with which users can authenticate each other.

Figure 5.8 illustrates how the Veno2 gateway cooperates with the Veno2 agent (VenoSec

software) installed in remote user's host to provide secure forwarding services. Observing the figure, the whole forwarding process consists of four phases: negotiation, authentication, channel association, and secure data relay.

Figure 5.8 Secure forwarding through Veno2 gateway

Initially, the negotiation phase is conducted between the agent and the gateway, which negotiates protocol version and agrees upon a common set of security mechanisms, such as authentication algorithm, cipher algorithm and integrity hash algorithm. Then the agent and the gateway conduct the authentication phase to authenticate each other through the just negotiated authentication algorithm. After successful authentication, a secure channel is created between the agent and the gateway. Then the gateway creates a channel with the destination server on behalf of the agent, and associates this channel with the security channel on the left side.

In the successive secure data relay phase, the gateway securely relays subsequent data transferred between the agent and the gateway. Specifically, the agent in the remote user side intercepts the application data to be sent and encrypts the data with the encryption algorithm negotiated in previous negotiation phase. Then it transmits the encrypted data to the gateway. The gateway decrypts the data and relays the original application data to the destination server.

Veno2 agent can discover the address information of the Veno2 gateway of an organization through a DNS prefixing mechanism. This technique obtains the name of the Veno2 gateway for a domain by adding a special DNS prefix (i.e. venossec) to the domain name, and resolves the name to gain the gateway's address information [76]. For

example, if Veno2 agent attempts to access the server `www.google.com`, then the address of the Veno2 gateway for this server is the correspondent address of DNS name `venossec.google.com`.

Considering the immense computational load being placed upon the Veno2 gateway, the performance of the gateway may be a problem. Administrator users in an organization can improve the performance by deploying multiple Veno2 gateways for load balancing. Besides, they can additionally install Veno2 agent (VenoSec software) in selected servers, which results in secure channel being directly established between client and these servers, hence relieve the burden in Veno2 gateway.

5.1.5 Experiment and Analysis

In this section, we present experiment results obtained from both testbed and real Internet. The results have three parts of it. First, we validate the functionality of VenoSec by employing VenoSec to enhance security to various popular user applications. Second, we analyze VenoSec and discuss its features. And then, we evaluate the performance degradation incurred by VenoSec, and demonstrate that the performance is acceptable for practical use.

5.1.5.1 Functions Testing

To validate the functionality, we employ VenoSec with various existing software in real Internet environments. In this section, we present the experiment results and discuss the problems we met and their solutions.

We conduct experiments for two types of applications: 1) traditional client-server applications, such as Telnet, Ftp and Http; 2) client-server-client applications, where two

clients communicate with each other through one or more intermediate servers, e.g. Instant Message (MSN, ICQ, QQ, Yahoo messenger) and Email (SMTP, Pop3, Webmail).

For the client-server applications, we deploy VenoSec with SSL module in both client and server machines. The SSL modules on both sides are responsible to authenticate each other and encrypt data transmitted between them. Specifically, we test the following software:

1. Telnet: Windows native Telnet client and server.
2. Ftp: Windows native Ftp client and server.
3. Http: Internet Explorer, Windows Web server.

We find that most of the software works well with VenoSec except for the Web server in Windows 2003. This is because Web server is implemented within the operating system kernel in Windows 2003. As VenoSec works in the application layer, it cannot intercept the network activities from the Web server and hence is unable to work properly. Our solution to this problem is to implement a simple stub driver to be loaded into Windows 2003 kernel, which can redirect web server data to pass through the VS-Agent, just like what VS-Stub can do. In order to intercept network activities from Web server in kernel, the stub driver is implemented with TDI (Transport Driver Interface) [112] instead of with popular NDIS (Network Driver Interface Specification) [113], for the latter cannot obtain process name, hence being not able to capture only the network activities from Web server [114].

For the client-server-client applications, we test popular software and applications for both Instant Message and Email. VenoSec is deployed in two clients' machines, which simply use pre-shared key to encrypt chat message or email content between end users. Specifically, we test the following software and applications:

1. Instant Message: MSN, ICQ, QQ, and Yahoo messenger. The tested functionalities include normal chat, group chat and file transferring.

2. Email: Outlook Express and Outlook 2003 with SMTP and Pop3, Webmail for hotmail (www.hotmail.com).

We find that VenoSec works well for most of the software and applications. However, VenoSec cannot work for Webmail which already uses Https to encrypt data between the client browser and Webmail server. In such scenario, client users may desire further security by encrypting the email content end-to-end between the sender and the receiver. But, as the data is already encrypted by browser software with a public key from Webmail server, VenoSec is not able to recognize the encrypted data and hence not able to further encrypt the encapsulated email content. Our solution to this problem is to additionally bypass the SSL API in client browser software, and redirect the original plain text data to our processing module, which is then responsible for encrypting the email content with end-to-end shared key and conducting actual SSL communication with the Webmail server on behalf of the browser.

5.1.5.2 Analysis

VenoSec is able to provide security solution for various communication models. As shown in Figure 5.9, client host C1 and server host S1 may communicate with each other using many client-server applications, such as Telnet, HTTP, FTP and etc. These client-server communications can be protected, as soon as the VenoSec software is installed in client C1 and server S1. Also, two client hosts, C2 and C3, may communicate with each other through intermediate server S2, and the communication may involve many client-server-client applications, such as Instant Message (MSN, QQ, ICQ) and Email (SMTP, POP3, Webmail). In order to protect such communication, only the two client hosts (C2 and C3) need to install the VenoSec software, and no need to make any modification in intermediate server S2.

Figure 5.9 Applying Venosec in two communication models

Moreover, Venosec can be deployed in various network scenarios. As being illustrated in Figure 5.10, communications can take place in three scenarios: 1) between individual client C1 and individual server S1; 2) between individual client C2 and its enterprise's intranet N2; and 3) between two enterprises' intranets N3 and N4. For the first scenario, Venosec is installed in both individual hosts (C1 and S1) to provide secure channel between them. To protect the communication in second scenario, a veno2 gateway is deployed in N2 and Venosec software is installed in C2. Moreover, two enterprises (N3 and N4) can communicate with each other securely, as soon as they both deploy a veno2 gateway in their intranets and install Venosec software in their client hosts.

Figure 5.10 Applying Venosec in three network scenarios

In summary, we evaluate Venosec based on the five main criteria as mentioned in Section 2.1.2: *security services*, *applicability*, *transparency*, *compatibility* and *deployability*.

Firstly, VenoSec can provide various security services, such as authentication, integrity, confidentiality and etc. in a configurable and extensible way. Also, VenoSec has good applicability as it can protect various applications no matter they are client-server or client-server-client model, or they are TCP or UDP. In addition, because VenoSec employs API interception technique, it is transparent to both legacy application software and operating system. Moreover, as working in application layer, VenoSec is compatible with existing network layer protocols and middle-boxes: such as NAT, VPN, and firewall. Finally, deployability is the salient feature of VenoSec, as it can protect various applications in diverse network scenarios.

5.1.5.2.1 VenoSec Integrate Security Techniques and Applications

The fundamental feature of VenoSec is its ability to integrate various security techniques/algorithms into a specific application without any modification to the application program.

A lot of application programs have no or very weak security protection. This is because most traditional applications were not developed with security in mind and many new emerging applications do not consider security in their initial design. As a result, attackers can easily compromise these applications through either active or passive attacks, such as sniffer, password recovery, man-in-the-middle, and etc.

To protect an application, the typical solution is to apply one security technique to the application, which requires modifying the existing application program to be equipped with the security technique (as depicted in Figure 5.11(a)). Such solution is much like integrating a key into the application program, so that attackers (who don't have the key) are not able to compromise the application. However, it is quite difficult to apply this solution for all applications considering the huge variety of them (which is like cutting a key for every application). Moreover, every time when application wants to use another

security technique or upgrade to a new one, the correspondent application program needs to be modified again (which is like cutting a new key for the application).

Figure 5.11 VenoSec works like a keyhole

Being different from the above-mentioned solution, VenoSec can transparently apply various security techniques into a specific application without any modification to the application program. This is much like cutting a generic keyhole in the application program, into which various keys (security techniques) can be inserted (as shown in Figure 5.11(b)). In this way, a common security technique can be easily applied to different applications, and also an application can easily change to use another security technique or upgrade to a new one (which is like inserting another key into the keyhole). In a word, VenoSec makes it more convenient to integrate security techniques and applications.

5.1.5.2.2 Applying VenoSec to Protect Against Realistic Attacks

There are many different kinds of network attacks over the Internet, such as eavesdropping, data modification, identity spoofing, password recovery and etc. Each of these network attacks has correspondent countermeasures which were proposed by security researchers. In fact, an application is usually vulnerable to more than one kind of

attacks. Consequently, to protect the application requires deploying multiple security techniques. This usually involves great effort to modify the application programs. Also, after new attacks are discovered, the application programs should all be updated with correspondent new security countermeasures.

Figure 5.12 VenoSec bridge the gap between security techniques and applications

By bridging the gap between security techniques and applications, VenoSec can facilitate the deployment of security countermeasures against various attacks. As illustrated in Figure 5.12, VenoSec is able to dynamically integrate appropriate security techniques into different application according to the application requirements and user configuration. For example, to protect Telnet application, we can apply VenoSec with PKI authentication, Diff-Hellman key exchange, 3DES encryption/decryption and MD5 integrity check. Also note that the Telnet program itself does not need any modification, thus the same protection can be applied to different Telnet programs. Since the existing programs do not need any modification, users can continue using their favorite software and their daily habit is well kept. Moreover, when a security technique is later found to have flaw [119][120][121] or a new stronger technique is proposed [122][123][124], it's much easier to just modify the configuration of VenoSec to use the new technique than to

modify the implementation of all correspondent programs.

Network Attacks	Security Services	Recommended Security Techniques
Eavesdropping	Confidentiality	3DES, AES, RC4
Traffic analysis	Confidentiality	3DES, AES, RC4
Identity spoofing	Authentication/ Non-repudiation	PKI, PGP
Man-in-the-middle	Authentication	PKI, Kerberos
Password recovery	Authentication	RSA, DSA
Data modification	Integrity	SHA
Replay	Integrity	SHA, Key refreshing
Denial of service	Availability	Load balance/Redundancy

Table 5-3 Recommended security techniques

Considering there are many network attacks and each attack has various countermeasures, we list several main security attacks and our recommendation of countermeasures against each of those attacks, as shown in Table 5-3.

5.1.5.2.3 Two Examples

In this section, we present two examples to show how to apply VenoSec to provide three basic security services: authentication, confidentiality and integrity.

Authentication

Authentication indicates a sender and receiver can confirm each other's identity. An attacker can compromise an authentication system by impersonating as the sender/receiver through mechanisms like password discovery, man-in-the-middle, session hijack and etc. To counter these attacks, many security techniques [36][116][117][118]

have been proposed, which can provide strong authentication services.

However, in order to utilize these strong authentication services, existing applications must be modified, either using specific implementations or existing libraries, both are not practical considered the great amount of applications. In this respect, VenoSec can be viewed as a complement of these specific security techniques. In other words, VenoSec can transparently integrate a specific security technique into an existing application without any modification.

Moreover, VenoSec provides an Authentication Provider Interface, which defines a common interface to integrate existing and new emerging security techniques in a generic fashion. As a result, third parties can integrate their own authentication mechanisms/protocols into popular applications conveniently. Also, when an authentication technique is later found to have flaw [119][120][121] or a new stronger technique is proposed [122][123][124], VenoSec can easily switch to the new technique by simply configure the Authentication Provider Interface to use the new one.

Figure 5.13 Applying VenoSec for strong authentication

For example (as shown in Figure 5.13), suppose the employees in an enterprise access the enterprise's email server using SMTP protocol, which authenticates through insecure plain-text password. And now the network manager of the enterprise plans to enhance the

email accessing by strong authentication. He can choose from many authentication techniques, such as Kerberos, PKI, PGP and etc. No matter which one he chooses, however, he needs to rewrite the current email client and server applications to support the new authentication technique, which involves great time and effort. With VenoSec, he can simply deploy the VenoSec software in client and server hosts, without modifying the current email applications. Moreover, if the enterprise wants to deploy an alternative authentication technique in the future, the network manager only needs to configure the VenoSec software to use the new technique.

Confidentiality and Integrity

Confidentiality or privacy keeps information from being exposed to anyone for whom it was unintended; and integrity enables a receiver to make sure that the data s/he receives is the true data without any alternation in transit. As most traditional applications were not developed with security in mind and many new emerging applications do not consider security in their initial design, yet, a lot of applications are still transmitting sensitive data in plain-text. As a result, attackers can easily compromise the confidentiality by eavesdropping and tamper the integrity by intercepting and modifying in-transit messages as their like. To counter these attacks, both private key techniques [125][126][127][128] and public key techniques [129][130][131] have been proposed.

Similar to authentication, VenoSec can also be viewed as a complement of these confidentiality and integrity techniques, as it is able to transparently integrate these techniques into existing applications without any modification to them. Also, when better confidentiality and integrity techniques are available [132][133][134], VenoSec can easily upgrade to use the new ones through the Data Protection Provider, which provides a generic interface to various cryptography algorithms/protocols.

Figure 5.14 Applying VenoSec for confidentiality and integrity

As shown in Figure 5.14, for example, most of the popular Instant Message systems, like MSN, ICQ, and Yahoo Messenger, still exchange users' private messages in plain-text. As a result, attackers can easily sniff the network to read all the private messages and even modify them by manipulating intermediate gateways or routers. VenoSec provides a convenient solution to these attacks by transparently encrypting/decrypting the exchanged private messages. Since the existing applications do not need any modification, users can continue using their favorite software and their daily habit is well kept.

5.1.5.2.4 Weaknesses

When VenoSec is deployed in an enterprise network, the communication path inside the enterprise network is not fully protected if the host within the enterprise has not installed VenoSec software. As illustrated in Figure 5.15(a), suppose VenoSec is deployed in client Host H1 and Gateway VG1. In this case, the communication between C1 and VG1 is protected, but the communication path between VG1 and S1 is still vulnerable to internal attacks inside the enterprise. Also, suppose VenoSec is deployed in Gateways VG2 and VG3 in Figure 5.15 (b). Then the Internet traffic between the two enterprises is protected, but communication paths within the enterprises (between C2 and VG2, VG3 and S3) are still not protected.

(a) Individual user access enterprise server

(b) Between two enterprises

Figure 5.15 Weaknesses

But, hosts within an organization are normally considered trustable by network manager, and thus the communications between them usually need not to be protected. In this case, VenoSec is consistent with typical network security requirements of an organization. However, if network manager deems it is necessary to further provide security for traffic within the organization, he can achieve this by incrementally deploying VenoSec in the hosts inside.

5.1.5.3 Performance

Adding security services into an application usually causes certain performance degradation. VenoSec affects the performance of an application in two aspects: 1) the architectural overhead from the VenoSec layer inserted into the target application, involving additional buffering, queuing and thread synchronization; and 2) the processing overhead resulting from the security protocol and cryptography algorithms. In the following, we present the benchmarking results to evaluate these two performance overheads. Of course, the performance tests only aim to show that the performance after employing VenoSec is still acceptable. The real benefit of VenoSec should be measured by its ability to transparently protect applications as discussed in Section 5.1.5.1.

Performance Impact of Veno2 Agent

We evaluate the performance impact of Veno2 agent by two benchmarks: bandwidth and latency. In bandwidth test, we use the Iperf [105] program to measure the throughput between a TCP client and a TCP server. In latency test, we measure the round trip time, which is the time for a TCP client to send a single byte to a server till it receives one byte response from the server.

Figure 5.16 Testing environments

We test bandwidth and latency benchmarks in the following three environments (as shown in Figure 5.16):

1. Local host. Both client and server programs are running in a same machine and communicate through loopback interface. The machine is equipped with 3.4 GHz Pentium CPU and 1 GB RAM running Windows XP professional (Service Pack 2).
2. LAN. The client and server are connected via a 100 Mbps LAN. The server is running in a 3.4 GHz Pentium machine with 1 GB RAM running Windows Server 2003 (Service Pack 1).
3. WAN. The client locates in NTU (Nanyang Technological University) in

Singapore and the server is in FDU (Fudan University) in China. The server is running in a 2.4 GHz Xeon machine with 2 GB RAM, installed with Linux Ubuntu Server 8.10.

First, we install Veno2 agent software in the client host for evaluating its performance impact to the machine itself. Noted we does not use VenoSink or VenoSec here in order to eliminate the processing overhead, which may be caused by the IPv4/IPv6 transition function or security function.

Figure 5.17 Veno2 bandwidth results in local host

Figure 5.17 reports the results of bandwidth in local host environment. Each point in the figure indicates the throughput for client to send data to server under a specific send block size. Comparing to the throughput without the Veno2 agent installation, seeing figure, we can get that the throughput performance of Veno2 is degraded almost by half when the send block size is large. This is because in local host environment, the bottleneck is mainly due to the processing time of the machine. Specifically, Veno2 needs two more sockets assisting to send data to server, hence causing the throughput to degrade.

Figure 5.18 Veno2 bandwidth results in LAN and WAN

Secondly, we evaluate Veno2 in LAN and WAN environments. Figure 5.18 illustrates that Veno2 introduces no or little performance penalty for large send block size, and it performs even better than native connection in small block size. This is because Veno2 uses 64 KB for both sending block size and receiving block size. Thus, even application programs send data in small block size, yet Veno2 will buffer the data and finally send out with large block size, resulting in higher throughput.

The above bandwidth test results indicate that Veno2 agent does incur performance degradation in local host environment. However, in network environments, such as LAN or WAN, Veno2 agent does not degrade performance, instead, it even improves performance somehow.

Figure 5.19 Veno2 latency results

In Figure 5.19, we have latency tests for all three cases. It is shown that Veno2 agent incurs a larger latency (e.g., several hundred microseconds) in local host and LAN environments. However, since the absolute value of latency in these environments is very small, thus this overall latency is acceptable. Similarly, we also test latency in Internet environment, Figure 5.19 shows that latency increment caused by Veno2 agent is very small.

Performance Impact of VenoSec

Besides above discussion, VenoSec may incur additional performance degradation caused by security processing like encryption/decryption or computing secure hash value. In order to evaluate the performance degradation, we deploy VenoSec in both client and server machines, and VenoSec will employ SSL to protect the communication.

Figure 5.20 VenoSec bandwidth results in local host

Figure 5.20 reports the bandwidth results in local host environment. It indicates that VenoSec reduces the throughput to around 110Mbps. This degradation is because of the inevitable security processing by SSL, such as encryption/decryption, computing secure hash value and compressing.

Figure 5.21 VenoSec bandwidth results in LAN

In LAN environment, with large send block size, VenoSec incurs no or little performance degradation, as depicted in Figure 5.21. This is consistent with our expectation because now the SSL processing is no longer the bottleneck. For small block size, VenoSec may achieve better performance than native connection since it internally uses a larger block size to send and receive data.

The above bandwidth results show that although VenoSec causes peak throughput to reduce in local host environment, yet in network environments, such as LAN or WAN, VenoSec incurs no or only little performance degradation.

Figure 5.22 VenoSec latency results

The latency results are shown in Figure 5.22. In both local host and LAN environments,

VenoSec increases the latency by about 800 microseconds. However, since the absolute value of latency in these environments is very small, thus this overall latency is acceptable. In Internet environment, network latency is usually tens or hundreds of milliseconds, hence we believe the several hundred microseconds latency caused by VenoSec is very small.

5.2 VenoSink: Instance of Veno2 for IPv4/IPv6 Transition Platform

5.2.1 Overview

Today, IPv6 backbones are widely deployed around the world. Being independent of IPv4 backbones, IPv6 backbones, i.e., Internet2 [23] in US, EURO6IX [24] in Europe, WIDE [25] in Japan, 6NGIX [26] in Korea, and CNGI [27] in China, are usually built up as separate infrastructures. However, currently there are very few IPv6 applications running over them. As compared to IPv4 which is always hungry for the bandwidth, IPv6 bandwidth is totally under-utilized because of lack of applications. With our proposed Veno2 transition architecture, data/streaming traffic will be spread in both IPv4 and IPv6 networks and such an unbalanced situation will be changed significantly. More importantly, such architecture will speed up the transition of IPv4 networks to IPv6 while there are no changes required on existing IPv4 and/or IPv6 infrastructures.

Researchers have attempted many different transition techniques over last ten years. Dual stack and tunneling techniques are extensively studied to provide the IPv6 connectivity between two separate IPv6 nodes/sites. NAT-PT [6], TRT [7] and SOCKS64 [8] are only gateway techniques designed to provide network translation between IPv4 and IPv6 networks, they do not take into account of applications dependence on IPv4/v6 layer. BIS

[30] and BIA [31] are proposed to migrate IPv4 applications to directly communicate with some of IPv6 applications or vice versa. However, considering practical limitations and specific situations, all these techniques are not widely used.

All these piecewise techniques have not been deployed on a large scale because of their applicability of only specific scenarios. As a matter of fact, users or ISPs have no interest to adopt IPv6 because of no obvious benefits immediately from deploying IPv6, furthermore, their daily use of IPv4 resource (i.e., MSN, QQ, Games, Email, WWW) may be more or less disrupted and make them quite inconvenient. That is the fundamental reason why today the number of IPv6 users are quite behind expected and IPv6 bandwidth is scarcely used all the time in past few years. Up to date, no solution architecture exists to provide transition between IPv4 and IPv6 without disruption of IPv4 resources. In this section, we propose VenoSink aiming to make users/ISPs enjoy benefits directly from IPv6 while there is no disruption to their IPv4 daily services. There are no changes required on existing IPv4 and/or IPv6 infrastructures, the only need is to just deploy a Veno2 *gateway* inside an enterprise network and install a Veno2 *agent* for each client node. In the following, we will explain how it works in details.

5.2.2 VenoSink Mechanism

Figure 5.23 is an enterprise network with one IPv4 network and one IPv6 network included. In IPv4 network, there are two nodes, one is a client Host C4 (155.69.1.1), another is a server Host S4 (155.69.1.1); similarly, in IPv6 network, a client Host C6 is configured with address of 1993:1::1, a server Host S6 is 1993:1::2. Veno2 transition architecture is quite simple. It is composed of Veno2 *gateway*, and Veno2 *agent*. *Gateway* is a Linux-based *forwarder*, deployed with dual stack and connected to IPv4 network and IPv6 network as well, seeing Figure 5.23. *Agent* is installed on each of client nodes in either IPv4 or IPv6 networks and can be launched by users in just one click.

Veno2 *gateway* is designed not only to provide a TCP or UDP flow relay between an IPv4 node and an IPv6 node, as such in SOCKS64, but also to provide an alternative flow path if source and destination nodes are both connected to IPv4 (or IPv6) networks.

Figure 5.23 An Enterprise Scenario with IPv4 network and IPv6 network included

The purpose of Veno2 *agent* is to 1) make application choose direct or indirect connectivity path based on DNS resolving result, and 2) thus migrate IPv4 applications to IPv6 applications or vice versa if it is necessary, this migration function is similar to BIA [31], but with many enhancements.

Observing Figure 5.23, let's illustrate how *agent* is invoked and how *gateway* is performed to realize the communications in four scenarios, 1) the communications between C4 and S4, 2) the communications between C6 and S6, 3) the communications between C4 and S6, 4) the communications between C6 and S4.

When IPv4 applications on the C6 with IPv6 stack communicate with a destination host, i.e. S6, S4, the *agent* will query DNS server to resolve the address of such requested destination host, if an AAAA record is available, the applications are allowed to communicate directly with this destination after the *agent* carries socket API mapping functions from IPv4 to IPv6. This behavior is very similar to that proposed in BIA [31].

However, if an A record is available, the applications have to communicate with this destination through the default Veno2 *gateway*. This gateway plays a role of relaying traffic from IPv6 networks to IPv4 networks and vice versa. Like above situation, in this case, the *agent* also needs to carry socket API mapping functions from IPv4 to IPv6 in the

local node. The only difference is that *agent* needs to set up a virtual communications channel between *agent* and *gateway*, then *gateway* and destination, thereafter, the data can be flown on this virtual channel. The function of Veno2 *gateway* here is much like a proxy's role, i.e., SOCKS64 [8].

Likewise, when IPv4 applications on the C4 with IPv4 stack communicate with a destination host, i.e. S6, S4, the *agent* will query DNS server to resolve the address of such requested destination host, if an A record is available, the applications are allowed to communicate directly with this destination as usual. There is no action for the *agent* this time.

If an AAAA record is available, the applications have to communicate with this destination through the default Veno2 *gateway*. This gateway plays a role of relaying traffic from IPv4 networks to IPv6 networks and vice versa. A virtual communications channel is set up between *agent* and *gateway*, then *gateway* and destination, thereafter, the data can be flown on this virtual channel. The function of Veno2 *gateway* here is much like a proxy's role, i.e., SOCKS64.

Considering that all applications are IPv4-dependent, with introduction of *agent* and *gateway*, IPv6 networks users may not be aware whether they are using IPv6 or not because their daily use habits are well kept. We test some existing software such as telnet, MSN, QQ, Outlook, Firefox, game and p2p applications. All those work. In fact, inside the enterprise networks, user can conveniently access all destination sites, unaware of they are IPv4 or IPv6. Of course, we also test IPv6 dependent applications, like telnet6, it works smoothly in above four scenarios without any disruption.

5.2.3 Technical Design

5.2.3.1 Agent

Veno2 *agent* is developed by inserting a thin layer in TCP/IP model. Figure 5.24 illustrates its location which is between the application layer and the socket layer.

Figure 5.24 Architecture of Veno2 agent for VenoSink

By this layer, all socket function calls by applications can be intercepted, processed and at last proceed to transport layer through the socket layer. To some extent, we can say *VenoSink* layer is kind of an extension of socket layer; the purpose is to provide more powerful session functions in the existing TCP/IP model.

Observing Figure 5.24, the VenoSink layer consists of three components: application transformer, path manager and Socket Switcher. In the following sections, we describe each of these components in details.

5.2.3.1.1 Application Transformer

BIA [30] and BIS [31] were proposed to migrate IPv4 applications to IPv6. However, these two application transforming techniques cannot migrate those applications, i.e., FTP, SNMP and SIP, which are embedded with IPv4 address. To solve this problem, ALG (Application Level Gateway) [6][97][98] is proposed to parse specific application layer protocol and thus replace IPv4 address (embedded in the application traffic) by a corresponding IPv6. Obviously, for each new application, a new dedicated ALG has to be rebuilt to handle.

Being different from above specific techniques, we propose Application Transformer, which aims to provide generic APIs to parse all application layer protocols and to replace IPv4 address embedded in the application traffic data. With these APIs, there is no need to develop specific ALGs for any new application protocols.

Seeing Figure 5.24, Application Transformer consists of two parts: Application Parser and Application Converter. The Application Parser provides an engine to generate parser for application parsing. In other words, a corresponding parser can be automatically created by a script.

The Application Converter provides callback functions to transform the IP information embedded in the application layer data. Specifically, when the Application Parser finds IPv4 address embedded in application data stream during parsing, callback functions registered by programmers will be invoked, transforming the retrieved IPv4 information to IPv6 or vice versa.

5.2.3.1.2 Path Manager

Looking at Figure 5.24, Path Manager locates in the middle part of the VenoSink layer, its function is to discover probable paths from a source to a destination, and select an optimal path among them. Essentially, it conducts application-level routing/switching,

and shares some commonalities with [55][99][61][100]. However, it is much simpler and more efficient, and integrates more closely with the Veno2 infrastructure. It consists of two components: Path Discover and Path Selector.

Path Discover

The responsibility of Path Discover is to discover all probable paths from a source to a destination. It has two functions: 1) discovering network connectivity information of source and destination hosts; 2) maintaining unconnected link state in a local cache.

Path Selector

Path Selector is responsible to select an optimal path from several paths found by Path Discover.

5.2.3.1.3 Socket Switcher

As shown in Figure 5.24, Socket Switcher locates in the bottom of the VenoSink layer. Its function is to switch from a high-level socket in network application to a low-level socket in socket layer. The high-level socket represents a communication endpoint of the network application. It specifies the peer endpoint of communication - the address of the destination. The low-level socket represents a specific route to the destination. It specifies how to reach the destination with or without relaying through intermediate default or external gateways. A high-level socket may have several corresponding low-level sockets. This is because, in VenoSink, a source and a destination may have several paths between them, and each path is represented by a low-level socket.

The Socket Switcher consists of 3 modules: Name Resolver, Address Mapper and Path Switcher.

Name Resolver

The function of Name Resolver is to return a proper answer in response to name resolving request from application, similar to [31]. For example, when application tries to resolve IPv4 address of a DNS name that only has corresponding IPv6 address, the Name Resolver requests the Address Mapper to assign a fake IPv4 address corresponding to the IPv6 address and returns the fake IPv4 address to the application.

Address Mapper

The Address Mapper maintains a mapping table that maps an IPv4 address to an IPv6 address or vice versa. It also assigns fake IPv4 or IPv6 addresses from an address pool (e.g. 0.0.0.1 ~ 0.255.255.255 for IPv4 and ::2 ~ ::255.255.255.255 for IPv6).

When the Name Resolver requests a fake IPv4 or IPv6 address, the Address Mapper needs to select an unused address from the address pool and mark this address as used [31]. Also, after each name resolution, the Address Mapper registers a mapping between resolved IPv4 addresses and resolved IPv6 addresses into the mapping table. This mapping enables the Path Manager to retrieve all the corresponding IPv4 and IPv6 addresses of a host, which is prerequisite information to discover paths.

Path Switcher

Path Switcher switches a high-level socket in network application to go through a specified path. We assume an application wants to access www.google.com and it creates a socket S_a to connect. When S_a proceeds to Veno2 agent, the Path Discover in Veno2 agent discovers 3 possible paths to reach the destination, and requests the Path Switcher to try the first path. Then the Path Switcher switches E_a (Endpoint Identifier of S_a) to Pa_1 (Path Identifier of the first path), and creates a new low-level socket S_{a1} in socket layer to connect to the destination. If this path fails, the Path Discover requests the Path

Switcher to try the second path. The Path Switcher then switches *Ea* to *Pa2* and creates socket *Sa2* to connect.

5.2.3.2 Gateway

Veno2 *gateway* is a server application running in a Linux/Unix machine with dual stack and connected to both IPv4 network and IPv6 network. A Veno2 *gateway* is usually deployed for an enterprise network who wishes to provide both IPv4 and IPv6 connectivity to the hosts inside. Specifically, Veno2 *gateway* plays two roles: data forwarder that forwards data between IPv4 network and IPv6 network, and DNS forwarder that forwards DNS request and response between IPv4 network and IPv6 network.

5.2.3.2.1 Data Forwarder

In the Veno2 architecture, we move routing complexity from intermediate nodes (Veno2 *gateways*) to edge nodes (Veno2 *agents*). Specifically, Veno2 *agent* is responsible for complex *routing* functions, such as discovering path, maintaining link state cache and selecting path; and Veno2 *gateway* only conducts simple *forwarding* function, which receives data from client and relays the data to its real destination. By this design, we make Veno2 *gateways* simpler and thus more efficient. Although Veno2 *agents* become more complex, yet current clients become more and more powerful and have enough capacity to handle such *routing* functions.

Data Forwarder of Veno2 *gateway* is responsible for forwarding data received from client to its real destination. It is similar to the function of a proxy, i.e. SOCKS64.

Figure 5.25 illustrates how the Data Forwarder in default *gateway* directly forwards data

from an IPv4 client to an IPv6 server. The forwarding consists of four phases: negotiation, authentication, channel association, and data relay.

Figure 5.25 Direct Forwarding by Data Forwarder

Initially, the negotiation phase is conducted between the client and the *gateway*, which negotiates protocol version and agrees upon an authentication method, with which the *gateway* can authenticate the client. Then the *gateway* conducts the authentication phase to verify that the client is a valid user. After successful authentication, the client tells the *gateway* the server's IPv6 address, and then the *gateway* creates an IPv6 channel from itself to the server and associates this channel with the client. Afterwards, the gateway relays any data received from one side to the other.

5.2.3.2.2 DNS Forwarder

The correct function of Veno2 relies on the reliability of DNS system. However, in real deployment of Veno2, we found DNS support is limited in IPv6 network. For instance, the DNS resolving library in Windows XP/2000/2003 does not support resolving through IPv6, thus many applications cannot work properly in pure IPv6 network environment under Windows XP/2000/2003. Besides, IPv4 DNS servers usually provide better service than IPv6 DNS servers, in terms of update frequency and amount of recognized names.

To tackle IPv6 DNS problems, we design Veno2 *gateway* to also work as a DNS forwarder, which can forward DNS request/response traffic between IPv4 network and

IPv6 network. By resolving through the DNS forwarder, even clients in pure IPv6 network can access IPv4 DNS servers reliably.

Specifically, we use the BIND [103][104] DNS server program in Linux/Unix as the DNS forwarder. A named server can be configured as a forwarder, with one or more DNS servers as its forwarding targets. We assume that each enterprise network has default IPv4 DNS servers. So we configure these IPv4 DNS servers as the forwarding targets of the named server. Consequently, when the name server in Veno2 gateway receives a client DNS request from IPv6 network, it forwards the request to one of IPv4 forwarding targets, and when receiving response from the IPv4 DNS server, it returns the response to the client by IPv6.

5.2.4 Experiment and Evaluation

The goal of VenoSink is to provide a transitional architecture for users to conveniently transit between IPv4 and IPv6 networks, without introducing excessive performance and deployment overhead. In this section, we present an evaluation of how VenoSink meets these goals. The evaluation contains two parts. First, we validate the functionality of VenoSink by demonstrating that it supports various popular user applications. Next, we test the performance of VenoSink on both testbed and real Internet environments, and conclude that the performance penalty caused by VenoSink is acceptable for practical use.

5.2.4.1 Functions Testing

In order to validate the functionality of VenoSink platform and implementation, we apply VenoSink to various applications in real Internet environments. In this section, we present results from our experiments, and discuss several problems and our solutions to them.

We conduct experiments in two types of environments: 1) homogeneous environment where IPv4 applications running in IPv4 network and IPv6 applications in IPv6 network. The purpose of this test is to ensure VenoSink does not affect normal communication; 2) heterogeneous environment where IPv4 (IPv6) applications running in IPv6 (IPv4) networks. This test aims to verify that VenoSink is able to transit application to utilize different underlying network. In this test either local Veno2 gateway or remote external gateway is configured.

We tested the following software in Windows platform:

1. Web browser: Internet Explorer, Firefox [106], Windows Web Server;
2. Email Client: Outlook Express, Outlook 2003;
3. Instant Message: MSN [32], ICQ [53], QQ [33].
4. Download tools: Windows FTP client, Cute ftp.

We find that most of the software works well after being employed with VenoSink in both homogeneous and heterogeneous environments. However, there are three problems: 1) DNS resolving library in Windows XP/2000/2003 does not support resolving through IPv6, thus many software using DNS name cannot work in pure IPv6 environment; 2) The Web server in Windows 2003 is implemented within the OS kernel [111], hence VenoSink cannot transit it; 3) Ftp software and file transferring in Instant message software cannot work properly.

Our solution to the first problem is to use our own DNS resolver and forwarder. As mentioned in Section 5.2.3.1.3 and Section 5.2.3.2.2, we employ three mechanisms: 1) we design Veno2 gateway to also work as a DNS forwarder, which can forward DNS request/response traffic between IPv4 network and IPv6 network; 2) the DNS Resolver is capable to resolve through IPv6; 3) the DNS Resolver can try multiple possible DNS servers to resolve name. With these mechanisms, the DNS Resolver can resolve names from the default Veno2 gateway as well as from native IPv4/IPv6 DNS servers.

For the second problem, we implement a simple stub driver to be loaded into Windows 2003 kernel, which can redirect web server data to pass through our processing module (VS-Agent). For more details, please refer to Section 5.1.5.1.

The solution to the third problem is through the Application Transformer, as described in Section 5.2.3.1.1. Currently, we implemented the application transformer for several popular Instant Message software, namely MSN, ICQ, Yahoo messenger and QQ, which modifies the IPv4/IPv6 address information in the correspondent application message, and can successfully transfer file over IPv4/IPv6 network.

5.2.4.2 Performance Evaluation in LAN

VenoSink works by inserting a Veno2 layer into target application and redirecting application network traffic to an alternative IPv4/IPv6 path (possibly through one or two Veno2 gateways). Thus, the performance degradation introduced by VenoSink consists of two parts: the architectural overhead from the inserted Veno2 layer, and the overhead of going through the alternative IPv4/IPv6 path. The former one is already discussed in Section 5.1.5.3. In the following, we focus on the overall performance degradation caused by VenoSink.

Figure 5.26 Four testing scenarios

Similar as evaluating performance of VenoSec, we use two benchmarks: the bandwidth by using Iperf [105] program and the latency by measuring the round trip time. Also, the test machines are the same as those in testing VenoSec. VenoSink may incur additional performance degradation caused by redirecting network traffic to go through an alternative path, which may traverse one or two Veno2 gateways. In the following, we compare the benchmark results for four scenarios (as shown in Figure 5.26): 1) direct communication between client and server without using VenoSink, 2) direct communication using VenoSink, 3) communication through one Veno2 gateway, and 4) communication through two Veno2 gateways.

Figure 5.27 VenoSink bandwidth results in LAN

Figure 5.27 indicates that VenoSink incurs only 5% and 10% of peak throughput reduction with one and two Veno2 intermediate gateways, comparing to the direct communication. This performance degradation is attributed to the additional buffering and queuing in the intermediate Veno2 gateways. For very small send block size, VenoSink performs even better. This is because VenoSink uses a larger block size (64KB) for sending and receiving data.

Figure 5.28 VenoSink latency results in LAN

Figure 5.28 shows latency test results in LAN environment. It indicates that VenoSink increases the latency by 600 and 700 microseconds with one and two gateways, comparing to direct communication. However, such increase of latency is not only caused by VenoSink processing, but also the longer alternative path through intermediate gateways. After eliminating the affect of path latency, we find that the actual overhead introduced by VenoSink is only about 400 microseconds, in which 300 microseconds by Veno2 agent, and 60 microseconds by each Veno2 gateway. We consider such small increase of delay is acceptable for practical use.

5.2.4.3 Performance Evaluation in WAN

We also conducted experiments in WAN environments. The testing mechanism and testing environments are similar to those in last section (as shown in Figure 5.26).

Figure 5.29 VenoSink bandwidth results in WAN

For the throughput metric, VenoSink does not cause bandwidth performance to degrade for large send block size, as depicted in Figure 5.29. Also, it improves bandwidth performance for small send block size.

Figure 5.30 shows the latency result in WAN. In fact, VenoSink only increases the delay by less than 1 millisecond. Such small increase can be neglected in real Internet communication, where network latency is usually tens or hundreds of milliseconds.

Figure 5.30 VenoSink latency results in WAN

In summary, we evaluate VenoSink based on the five main criteria as mentioned in Section 2.2.2: *robustness*, *compatibility*, *deployability*, *scalability* and *efficiency*. Firstly, VenoSink has good robustness as it avoids single point failure through deploying multiple

gateways. Also, as it works in application layer, VenoSink is compatible with existing network layer protocols and middle-boxes, such as NAT, VPN and firewall. In addition, because VenoSink employs API interception technique, it is easy to be deployed without modifying existing network infrastructure and legacy application software. Moreover, VenoSink is designed with load-balance in mind, and it can support site-to-site communication without introducing any change to routing table, thus we believe it has good scalability. Finally, VenoSink can achieve good efficiency through fast gateway discovery and smart routing selection.

Chapter 6

Conclusion and Future Work

To conclude this thesis, in Section 6.1 we briefly summarize our research work. Then Section 6.2 discusses directions for future work.

6.1 Summary

This thesis proposed a generic overlay infrastructure called Veno2, aiming to universally provide various session level services for all applications in a deployable fashion. Veno2 infrastructure is extensible as new session level services can be easily accommodated into it, and is flexible through configuring and organizing different services according to users' requirements. In addition, we enhance the reliability of Veno2 infrastructure with techniques like DNS forwarding, topology discovering. Moreover, the efficiency of the infrastructure is improved by balancing the processing complexities being put on Veno2 agent and Veno2 gateway.

To verify the idea of Veno2 infrastructure, we apply it to the security and connectivity issues, and design and implement the VenoSec platform and VenoSink platform.

In the security area, VenoSec platform aims to provide various security services for end users with very little deployment cost. In particular, VenoSec can be employed to protect new emerging client-server-client communications as well as traditional client-server

applications. Deployment of VenoSec can be achieved easily and incrementally in two typical scenarios: individual user scenario, where no central authority or agent is available, and group user scenario, where there is trusted authority and Veno2 gateway is deployed to assist secure data relaying. In order to provide an extensible and flexible architecture, the technical framework of VenoSec consists of modular components, namely application transformer, security policy manager and security provider interface.

VenoSink platform provides a transitional architecture between IPv4 and IPv6 networks. We investigate the usage of VenoSink for IPv4/IPv6 interaction within or between enterprise networks. VenoSink platform melts the border between IPv4 and IPv6 and merges two separate networks into a universal one, hence make the whole communication infrastructure more robust. For transiting applications with embedded IPv4 or IPv6 address information, an Application Transformer is provided for facilitating parsing application protocol and replacing embedded IPv4 address with IPv6, and vice versa.

Numerous experiments have been conducted in real networks and live Internet for both VenoSec and VenoSink platforms. The detailed investigations indicate that both architectures incur only little performance degradation, which is quite acceptable for typical network environments such as LAN or WAN. Meanwhile, functionality tests against various popular user applications show that VenoSec and VenoSink are capable of providing security and IPv4/IPv6 transition services respectively.

The salient feature of this Veno2 infrastructure is the great ease for deployment. Specifically, there are no changes required on existing Internet infrastructures, the only need is to just deploy a Veno2 gateway inside an organization's network and install Veno2 agent software in client node. Being different from techniques like VPN, NAT or firewall requiring the gateway server to be deployed at the edge of an organization's intranet, the Veno2 gateway can be deployed at any place of the Internet, hence enables more flexible deployment in different network environments. In addition, the deployment of Veno2 agent involves no modification to either application software or operating system kernel,

and even doesn't require administrative privilege.

6.2 Future Work

There are several interesting directions for future work.

- ◆ **Extend Veno2 infrastructure with other services, such as mobility, multi-homing.**

In this thesis, we have studied the application of the proposed Veno2 infrastructure in security and IPv4/IPv6 transition areas. In the future, we plan to extend the infrastructure to provide more services, such as mobility, and multi-homing. In particular, it is possible to realize mobility through Veno2 infrastructure by designing local and remote Veno2 gateways to work as home and foreign agents as in mobile IP [139][140]. To some extent, VenoSink has already provided certain kind of multi-homing by allowing fallback among multiple potential IPv4/IPv6 paths [141][142]. Future work should further explore more general problems related to multi-homing, such as the fallback within multiple IPv4 or IPv6 addresses.

- ◆ **Extend VenoSink to support more general form of transition in transport layer.**

Currently, VenoSink only conducts network layer transition between IPv4 and IPv6. In the future, we plan to extend VenoSink to support more general form of transition, which additionally includes transition between transport layer protocols, such as TCP, UDP, SCTP [143], and DCCP [144]. Actually, transition between TCP and UDP is already applied in applications like IPTV, VoIP and multicast. More comprehensive study and work are needed in the future to investigate various transition scenarios and integrate them into the Veno2 infrastructure.

◆ **Extend Veno2 to be a universal transport platform for next generation network.**

There are three clear evolutions in today's Internet as shown in Figure 6.1: 1) in application layer, voice and video transmission over TCP/IP are becoming increasingly popular; 2) in network layer, the uniformity of IPv4 has been broken by IPv6; and 3) in data-link and physical layers, wireless links play a more and more important role. These evolutions embody the deficiencies in original Internet design and ask for a new architecture.

Figure 6.1 Architecture of Universal Transport Platform

To meet the requirement for a new architecture, the future work of our team will focus on a universal transport platform in the transport layer, which can hide network heterogeneity in lower layers. This platform is based on TCP Veno [145] which solves the performance problem for TCP in wireless network. Through further study of transition between transport layer protocols, we plan to extend the Veno2 infrastructure to be such a universal transport platform, which will help evolve the current Internet to its next generation.

References

- [1] B. Leiner, V. Cerf, D. Clark, R. Kahn , L. Kleinrock, D. Lynch, J. Postel, L. Roberts, and S. Wolff, “The past and future history of the Internet”, Communications of the ACM, v.40 n.2, p.102-108, Feb. 1997.
- [2] J. Postel, “Internet Protocol: DARPA Internet Program Protocol Specification”, RFC 791, September 1981.
- [3] OECD document: <http://www.oecd.org/dataoecd/7/1/40605942.pdf>.
- [4] S. Deering and R. Hinden, “Internet Protocol, Version 6 (IPv6) Specification”, RFC 2460, December 1998.
- [5] B. Carpenter and K. Moore, “Connection of IPv6 Domains via IPv4 Clouds”, RFC 3056, February 2001.
- [6] G. Tsirtsis and P. Srisuresh, “Network Address Translation - Protocol Translation (NAT-PT)”, RFC 2766, February 2000.
- [7] J. Hagino and K. Yamamoto, “An IPv6-to-IPv4 Transport Relay Translator”, RFC 3142, June 2001.
- [8] H. Kitamura, “A SOCKS-based IPv6/IPv4 Gateway Mechanism”, RFC 3089, April 2001.
- [9] Eichin, M. and J. Rochlis, “With Microscope and Tweezers: An Analysis of the Internet Virus of November 1988”, Proceedings of the 1989 IEEE Symposium on Security and Privacy (Oakland CA, May 1989), IEEE Computer Society, 326-344.
- [10] T. Ylonen and C. Lonvick, Ed., “The Secure Shell (SSH) Protocol Architecture”, RFC 4251, January 2006.
- [11] SET Secure Electronic Transaction, “Book 1: Business Description”, “Book 2: Programmer's Guide”, “Book 3: Formal Protocol Definition”, May 31, 1997.

- [12] R. Arends, R. Austein, M. Larson, D. Massey, and S. Rose, "DNS Security Introduction and Requirement", RFC 4033, March 2005.
- [13] S. Kent and R. Atkinson, "Security Architecture for the Internet Protocol", RFC 2401, November 1998.
- [14] T. Dierks and C. Allen, "The TLS Protocol Version 1.0", RFC 2246, January 1999.
- [15] Leech, M., Ganis, M., Lee, Y., Kuris, R., Koblas, D. and L. Jones, "SOCKS Protocol V5", RFC 1928, April 1996.
- [16] E. Rescorla, "HTTP Over TLS", RFC 2818, May 2000.
- [17] B. Gleeson, A. Lin, J. Heinanen, G. Armitage and A. Malis, "A Framework for IP Based Virtual Private Networks", RFC 2764, February 2000.
- [18] ITU-T NGN Study Group: <http://www.itu.int/ITU-T>.
- [19] NGI mirror: <http://web.archive.org/web/20050916083130/http://www.ngi.gov/>.
- [20] L. Peterson, T. Anderson, D. Blumenthal, D. Casey, D. Clark, D. Estrin, J. Evans, D. Raychaudhuri, M. Reiter, J. Rexford, S. Shenker, and J. Wroclawski, "GENI design principles", IEEE Computer Magazine, 2006.
- [21] Steven M. Bellovin, David D. Clark, Adrian Perrig, Dawn Song (Eds), "Report of NSF Workshop on A Clean-Slate Design for the Next-Generation Secure Internet," GENI Design Document 05-05, July 2005.
- [22] AGN website: <http://www.darpa.mil/sto/solicitations/AGN/index.html>.
- [23] Debra Cameron, "Internet 2: The Future of the Internet and Next-Generation Initiatives", Computer Technology Research Corp., Charleston, SC, 1999.
- [24] Euro6IX (European IPv6 Internet Exchanges Backbone): <http://www.euro6ix.org>.
- [25] WIDE (Widely Integrated Distributed Environment): <http://www.wide.ad.jp>.
- [26] 6NGIX (IPv6 Next Generation Internet eXchange): <http://www.ngix.ne.kr>.
- [27] CNGI (China Next Generation Internet): <http://www.edu.cn/html/cngi>.
- [28] Nordmark, E. and R. Gilligan, "Basic Transition Mechanisms for IPv6 Hosts and Routers", RFC 4213, October 2005.
- [29] Zhu, Yuncheng Chen, Maoke Zhang, Hong Li, Xing, "Stateless Mapping and Multiplexing of IPv4 Addresses in Migration to IPv6 Internet", IEEE GLOBECOM 2008, New Orleans, LA USA, Nov. 30 2008-Dec. 4 2008.

- [30] K. Tsuchiya, H. Higuchi and Y. Atarashi, "Dual Stack Hosts using the "Bump-In-the-Stack" Technique (BIS)", RFC 2767, February 2000.
- [31] S. Lee, M-K. Shin, Y-J. Kim, E. Nordmark and A. Durand, "Dual Stack Hosts Using "Bump-in-the-API" (BIA)", RFC 3338, October 2002.
- [32] MSN messenger: [http:// get.live.com/messenger/overview](http://get.live.com/messenger/overview).
- [33] QQ messenger: [http:// im.qq.com](http://im.qq.com).
- [34] P. Srisuresh and M. Holdrege, "IP Network Address Translator (NAT) Terminology and Considerations", RFC 2663, August 1999.
- [35] B. Aboba and W. Dixon, "IPsec-Network Address Translation (NAT) Compatibility Requirements", RFC 3715, March 2004.
- [36] B. Clifford Neuman and Theodore Ts'o, "Kerberos: An Authentication Service for Computer Networks", IEEE Communications, 32(9) pp33–38. September 1994.
- [37] E. Nordmark, "Stateless IP/ICMP Translation Algorithm (SIIT)", RFC 2765, February 2000.
- [38] F. Templin, T. Gleeson, M. Talwar and D. Thaler, "Intra-Site Automatic Tunnel Addressing Protocol (ISATAP)", RFC 4214, October 2005.
- [39] C. Huitema, "Teredo: Tunneling IPv6 over UDP through Network Address Translations (NATs)", RFC 4380, February 2006.
- [40] A. Durand, P. Fasano, I. Guardini and D. Lento, "IPv6 Tunnel Broker", RFC 3053, January 2001.
- [41] J. Bound, L. Toutain, O. Medina, F. Dupont, H. Affifi and A. Durand, "Dual Stack Transition Mechanism (DSTM)", Internet-Draft draft-ietf-ngtrans-dstm-08, July 2002.
- [42] M-K. Shin, Ed., Y-G. Hong, J. Hagino, P. Savola and E. M. Castro, "Application Aspects of IPv6 Transition", RFC 4038, March 2005.
- [43] P. Ford-Hutchinson, "Securing FTP with TLS", RFC 4217, October 2005.
- [44] C. Newman, "Using TLS with IMAP, POP3 and ACAP", RFC 2595, June 1999.
- [45] OpenSSL website: <http://www.openssl.org>.
- [46] R. Shirey, "Internet Security Glossary", RFC 2828, May, 2000.
- [47] M. Leech, "Username/Password Authentication for SOCKS V5", RFC 1929, March 1996.

- [48] P. McMahon, “GSS-API Authentication Method for SOCKS Version 5”, RFC 1961, June 1996.
- [49] SocksCap download: <http://soft.softoogle.com/ap/sockscap-download-5157.shtml>.
- [50] J. Salz, A. Snoeren and H. Balakrishnan, “TESLA: A Transparent, Extensible Session-Layer Architecture for End-to-End Network Services”, 4th Usenix Symposium on Internet Technologies and Systems, Seattle, Washington, March 2003.
- [51] A. Zúquete and P. Guedes, “Transparent Authentication and Confidentiality for Stream Sockets”, IEEE Micro, 16(3):34–41, June 1996.
- [52] A. Zúquete and P. Guedes, “Transparent Authentication and Confidentiality for Datagram Sockets”, ERSADS '97, March 1997, Zinal, Switzerland.
- [53] ICQ messenger: <http://www.icq.com>.
- [54] C. P. Fu, “Session-Oriented Architecture Design in Mobile Internet”, Information Technology Workshop 2002, China.
- [55] D. Andersen, H. Balakrishnan, F. Kaashoek and R. Morris, “Resilient Overlay Networks”, in Proc. of SOSP, 2001.
- [56] Ratnasamy, S., Shenker, S., and McCanne, S., “Towards an evolvable internet architecture”, SIGCOMM Comput. Commun. Rev. 35, 4 (Oct. 2005), 313-324.
- [57] Larry Peterson, Scott Shenker, and Jonathan Turner, “Overcoming the Internet Impasse through Virtualization”, In Third Workshop on Hot Topics in Networks (HotNets-III), November 2004.
- [58] B. Cohen. Incentives build robustness in BitTorrent. In Workshop on Economics of Peer-to-Peer Systems, 2003.
- [59] I. Stoica, R. Morris, D. Karger, M. Kaashoek, H. Balakrishnan, “Chord: A scalable peer-to-peer lookup service for internet applications”, Proceedings of the 2001 conference on Applications, technologies, architectures, and protocols for computer communications, p.149-160, August 2001, San Diego, California, United States.
- [60] Huang, Y., Fu, T. Z., Chiu, D., Lui, J. C., and Huang, C., “Challenges, design and analysis of a large-scale p2p-vod system”, In Proceedings of the ACM SIGCOMM 2008 Conference on Data Communication (Seattle, WA, USA, August 17 - 22, 2008). SIGCOMM '08. ACM, New York, NY, 375-388.

- [61] SAVAGE, S., ANDERSON, T., ET AL. Detour: A Case for Informed Internet Routing and Transport. IEEE Micro 19, 1 (Jan. 1999), 50–59.
- [62] L. Subramanian, I. Stoica, H. Balakrishnan, and R. Katz, “OverQoS: An Overlay Based Architecture for Enhancing Internet QoS”, In Proc. of NSDI, 2004.
- [63] Z. Duan, Z. Zhang, and Y. T. Hou, “Bandwidth provisioning for service overlay networks”, in SPIEITCOM Scalability and Traffic Control in IP Networks (II), 2002.
- [64] J. Jannotti, D. K. Gifford, K. L. Johnson, M. F. Kaashoek, and J. J. W. O’Toole, “Overcast: Reliable multicasting with an overlay network”, in 4th USENIX OSDI, Oct. 2000.
- [65] S.Y. Shi and J.S. Turner, “Routing in overlay multicast networks”, in IEEE Infocom’02, June 2002.
- [66] A. D. Keromytis, V. Misra, and D. Rubenstein, “SOS: Secure Overlay Services”, In Proceedings of ACM SIGCOMM, pages 61–72, August 2002.
- [67] R. Rockell and R. Fink, “6Bone Backbone Routing Guidelines”, RFC 2772, February 2000.
- [68] Planetary Network Testbed, “<http://www.planet-lab.org>.”
- [69] R. Braynard, D. Kotic, A. Rodriguez, J. Chase, and A. Vahdat, “Opus: An overlay peer utility service”, in IEEE OpenArch’02, June 2002.
- [70] Touch, J, “Dynamic Internet overlay deployment and management using the X-Bone”, In Proceedings of the 2000 international Conference on Network Protocols (November 14 - 17, 2000). ICNP.
- [71] Joseph, D., Kannan, J., Kubota, A., Lakshminarayanan, K., Stoica, I., and Wehrle, K, “OCALA: an architecture for supporting legacy applications over overlays”, In Proceedings of the 3rd Conference on Networked Systems Design & Implementation - Volume 3 (San Jose, CA, May 08 - 10, 2006).
- [72] S. Kelly and S. Ramamoorthi, “Requirements for IPsec Remote Access Scenarios”, RFC 3457, January 2003.
- [73] D. Ritchie, “A Stream Input–Output System”, AT&T Bell Labs Technical Journal, vol. 63, pp. 311–324, Oct. 1984.

- [74] UNIX Software Operations, UNIX System V Release, 4, “Programmer’s Guide: STREAMS”, Prentice Hall, 1990.
- [75] Mockapetris, P., "Domain names - concepts and facilities", STD 13, RFC 1034, November 1987.
- [76] J. Palet, M. Diaz and P. Savola, “Analysis of IPv6 Tunnel End-point Discovery Mechanisms”, Internet-Draft draft-palet-v6ops-tun-auto-disc-03, January 24, 2005.
- [77] T. Brisco, “DNS Support for Load Balancing”, RFC 1794, April 1995.
- [78] Myers, D. S. and A. L. Bazinet, “Intercepting arbitrary functions on Windows, UNIX, and Macintosh OS X platforms”, UMIACS Technical Report CS-TR-4585, UMIACS-TR-2004-28.
- [79] Kdm, Kodmaker@syshell.org, “Analysis of a win32 userland rootkit”, Phrack62.
- [80] Robert Kuster, “Three Ways To Inject Your Code Into Another Process”, August 4, 2003, <http://www.codeproject.com/KB/threads/winspy.aspx>.
- [81] Jeffrey Richter, “Programming Applications for Microsoft Windows”, 4th Edition, Microsoft Press, 1999.
- [82] John Robbins, “Debugging Applications”, Microsoft Press, January 1, 2000.
- [83] Matt Pietrek, “Peering Inside the PE: A Tour of the Win32 Portable Executable File Format”, March 1994.
- [84] Matt Pietrek, “An In-Depth Look into the Win32 Portable Executable File Format”, Part 1 & Part 2, February & March 2002.
- [85] Galen Hunt and Doug Brubacher, “Detours: Binary Interception of Win32 Functions”, Proceedings of the 3rd USENIX Windows NT, Symposium. Seattle, WA, July 1999.
- [86] Lzik, “Reverse Engineering with LD_PRELOAD”, July 06 2005, <http://securityvulns.com/articles/reveng/>.
- [87] p59_0c@author.phrack.org, “Building ptrace injecting shellcodes”, Phrack59.
- [88] N. C. Hutchinson and L. L. Peterson, “The x-kernel: An Architecture for Implementing Network Protocols”, IEEE Transactions on Software Engineering, vol. 17, pp. 64–76, January 1991.
- [89] H. Miranda, A. Pinto and L. Rodrigues, “Appia, a flexible protocol kernel supporting multiple coordinated channels”, In Proceedings of The 21st Int’l Conf.

- on Distributed Computing Systems (ICDCS-21), pages 707–710, Phoenix, Arizona, USA, Apr.16–19 2001. IEEE Computer Society.
- [90] Matti A. Hiltunen and Richard D. Schlichting, “The Cactus Approach to Building Configurable Middleware Services”, In Proceedings of the Workshop on Dependable System Middleware and Group Communication (DSMGC 2000), Nuremberg, Germany (October 2000).
- [91] Sun Solaris White Paper, “FireEngine -- A New Networking Architecture for the Solaris Operating System”, November 2004.
- [92] Lesk, M. E., "Lex - a Lexical Analyzer Generator", Computing Science Technical Report 39, AT&T Bell Laboratories, Murray Hill, New Jersey, 1975.
- [93] Johnson, S. C., "YACC - Yet Another Compiler Compiler", Computing Science Technical Report 32, AT&T Bell Laboratories, Murray Hill, New Jersey, 1975.
- [94] N. Borisov, D. J. Brumley, H. J. Wang, J. Dunagan, P. Joshi, and C. Guo, “Generic application-level protocol analyzer and its language”, In 14th Annual Network & Distributed System Security Symposium, 2007.
- [95] R. Pang, V. Paxson, R. Sommer, and L. Peterson, “binpac: a yacc for writing application protocol parsers”, In Proceedings of the 6th ACM SIGCOMM conference on Internet measurement, October 25-27, 2006, Rio de Janeiro, Brazil.
- [96] PCRE (Perl Compatible Regular Expressions) library: [http:// www.pcre.org](http://www.pcre.org).
- [97] K. Yamamoto and M. Sumikawa, “Overview of Transition Techniques for IPv6-only to Talk to IPv4-only Communication”, Internet-Draft, draft-ietf-ngtrans-translator-03, March, 2000.
- [98] Chen, W.E., Su, C.Y. and Lin, Y.-B, “NCTU SLT: A Socket-layer Translator for IPv4-IPv6 Translation”, IEEE Communications Letters pdf , 9(10): 865-867, 2005.
- [99] P. Laskowski, B. Johnson and J. Chuang, “User-Directed Routing: From Theory, towards Practice”, Proceedings of ACM SIGCOMM Workshop on the Economics of Networks, Systems, and Computation (NetEcon 2008), August 2008.
- [100] R. Beyah, R. Sivakumar and J. Copeland, “Application layer switching: A deployable technique for providing quality of service”, In Proceedings of IEEE GLOBECOM Conference, 2003.

- [101] E. W. Dijkstra: A note on two problems in connexion with graphs. In *Numerische Mathematik*, 1 (1959), S. 269–271.
- [102] IPv6 in Windows: <http://www.microsoft.com/technet/network/ipv6/ipv6faq.mspix>.
- [103] BIND (Berkeley Internet Name Domain): <http://www.bind9.net>.
- [104] Douglas Brian Terry, Mark Painter, David W. Riggle and Songnian Zhou, “The Berkeley Internet Name Domain Server”, *Proceedings USENIX Summer Conference*, Salt Lake City, Utah, June 1984, pages 23–31.
- [105] Iperf website: <http://dast.nlanr.net/Projects/Iperf>.
- [106] Firefox website: <http://www.mozilla.com/firefox>.
- [107] Mathon website: <http://www.maxthon.com>.
- [108] Google Chrome website: <http://www.google.com/chrome>.
- [109] Opera website: www.opera.com.
- [110] Yahoo messenger: [http:// messenger.yahoo.com](http://messenger.yahoo.com).
- [111] George Shepherd, “IIS 6.0: New Features Improve Your Web Server's Performance, Reliability, and Scalability”, from issue of *MSDN Magazine*, March 2002, <http://msdn.microsoft.com/en-us/magazine/cc301710.aspx>.
- [112] TDI: <http://msdn.microsoft.com/en-us/library/ms801224.aspx>.
- [113] NDIS: <http://msdn.microsoft.com/en-us/library/bb961651.aspx>.
- [114] Oney, W., “Programming the Microsoft Windows Driver Model”, Second Edition. 2nd. Microsoft Press, 2002.
- [115] P. Srisuresh and M. Holdrege, “M. Holdrege”, RFC 2663, August 1999.
- [116] S. Chokhani and W. Ford, “Internet X.509 Public Key Infrastructure Certificate Policy and Certification Practices Framework”, RFC 2527, March 1999.
- [117] J. Callas, L. Donnerhake, H. Finney, D. Shaw, and R. Thayer, “OpenPGP Message Format”, RFC 4880, November 2007.
- [118] Rigney, C., Willens, S., Rubens, A., and W. Simpson, "Remote Authentication Dial In User Service (RADIUS)", RFC 2865, June 2000.
- [119] Tom Yu, Sam Hartman, and Ken Raeburn, “The Perils of Unauthenticated Encryption: Kerberos Version 4”, In *Proceedings of the Network and Distributed System Security Symposium*. The Internet Society, February 2004.

- [120] Cervesato, I., Jaggard, A. D., Scedrov, A., Tsay, J., and Walstad, C., “Breaking and fixing public-key Kerberos”, Inf. Comput. 206, 2-4 (Feb. 2008), 402-424.
- [121] Jallad, K., Katz, J., and Schneier, B., “Implementation of Chosen-Ciphertext Attacks against PGP and GnuPG”, In Proceedings of the 5th international Conference on information Security (September 30 - October 02, 2002).
- [122] Davis, D. and Geer, D. E., “Kerberos security with clocks adrift”, In Proceedings of the 5th Conference on USENIX UNIX Security Symposium - Volume 5 (Salt Lake City, Utah, June 05 - 07, 1995).
- [123] Hane, S., Fujishiro, T., Hashimoto, Y., Kaji, T., Katsuhiko, K., and Tezuka, S. “Speeding up X.509 Certificate Path Validation”, In Proceeding of the 2005 Conference on Applied Public Key infrastructure: 4th international Workshop: IWAP 2005.
- [124] Borisov, N., Goldberg, I., and Brewer, E. “Off-the-record communication, or, why not to use PGP”, In Proceedings of the 2004 ACM Workshop on Privacy in the Electronic Society (Washington DC, USA, October 28 - 28, 2004).
- [125] National Institute of Standards and Technology, “Data Encryption Standard”, FIPS. v46-3.
- [126] Announcing the ADVANCED ENCRYPTION STANDARD (AES). Federal Information Processing Standard FIPS 197, National Institute of Standards and Technology (NIST), Nov 2001.
- [127] R. Rivest, The MD5 Message-Digest Algorithm, RFC Editor, 1992.
- [128] B. Schneier, Applied Cryptography. New York: Wiley, 1994.
- [129] Ron Rivest, Adi Shamir, Leonard Adelman, "A Method for Obtaining Digital Signatures and Public-Key Cryptosystems", MIT Laboratory for Computer Science and Department of Mathematics. Communications of the ACM, February 1978, Volume 21, Number 2, pp. 120-126.
- [130] National Institute of Standards and Technology, “A proposed federal information processing standard for digital signature standard (DSS)”, Federal Register, vol. 56, no. 169, pp. 42,980-42,982, Aug. 30, 1991.
- [131] W. Diffie and M. E. Hellman, “New Directions in Cryptography”, IEEE Transactions on Information Theory, vol. IT-22, Nov. 1976, pp: 644–654.

- [132] Xiaoyun Wang, Denggno Feng, Xuejia Lai, Hongbo Yu, "Collisions for hash functions MD4, MD5, HAVAL- 128 and RIPEMD", Rump session of Crypto'04, Cryptology ePrint Archive, Report2004/199, 2004.
- [133] Dan Boneh, "Twenty Years of Attacks on the RSA Cryptosystem", Notices of the American Mathematical Society (AMS), Vol. 46, No. 2, pp. 203-213, 1999.
- [134] TripleDES, "Triple Data Encryption Algorithm Modes of Operation", ANSI X9.52, 1998.
- [135] Wendlandt, D., Andersen, D. G., and Perrig, A, "Perspectives: improving SSH-style host authentication with multi-path probing", In USENIX 2008 Annual Technical Conference on Annual Technical Conference (Boston, Massachusetts, June 22 - 27, 2008). USENIX Association, Berkeley, CA, 321-334, 2008..
- [136] American National Standards Institute. ANSI X9.52-1998, Triple Data Encryption Algorithm Modes of Operation, 1998.
- [137] National Institute of Standards and Technology. FIPS Pub 197: Advanced Encryption Standard (AES). 26 November, 2001.
- [138] J. Schaad and R. Housley, "Wrapping a Hashed Message Authentication Code (HMAC) key with a Triple-Data Encryption Standard (DES) Key or an Advanced Encryption Standard (AES) Key", RFC 3537, May 2003.
- [139] C. Perkins, Ed., "IP Mobility Support for IPv4", RFC 3220, January 2002.
- [140] D. Johnson, C. Perkins and J. Arkko, "Mobility Support in IPv6", RFC 3775, June 2004.
- [141] J. Abley, K. Lindqvist, E. Davies, B. Black and V. Gill, "IPv4 Multihoming Practices and Limitations", RFC 4116, July 2005.
- [142] E. Lear, "Things Multihoming in IPv6 (MULTI6) Developers Should Think About", RFC 4219, October 2005.
- [143] R. Stewart, Ed., "Stream Control Transmission Protocol", RFC 4960, September 2007.
- [144] E. Kohler, M. Handley and S. Floyd, "Datagram Congestion Control Protocol (DCCP)", RFC 4340, March 2006.
- [145] Fu, C.P. and Liew, S.C., "TCP Veno: TCP enhancement for transmission over wireless access networks", IEEE J. Select. Areas Commun. v21 i2. 216-228.