第5回ソフトウェア翻訳士認定試験

＜一次試験＞　3月6日（日）10：00～15：00

問題1、問題2の両方について解答のこと。選択ではありません。

問題1

下線部のみを和訳して提出してください。

Linux Network Security

Linux Firewall management

Regardless of your firewall type (proxy, packetfiltering, etc), it is not a good idea to have a firewall performing any more services than absolutely necessary. The services are best limited to the minimum services required to run the machine. I recommend that you do not provide NFS, TFTP, BOOTP, DHCP, web services, mail services, samba services, FTP, or telnet on your firewall unless absolutely necessary. If you must provide these services, be careful with wild cards in their configuration files that may allow blocks of systems or users to have access to your machine. Also if you are running these services, you should monitor security postings on these services so you are aware of any security holes associated with that particular service. If you must provide telnet or FTP, be sure you configure your tcp wrapper in the inetd.conf file for these services, and set the hosts.all and hosts.deny files to be as restrictive as possible. See the section on inetd services for information on how to do this. Policies for a firewall:

1. Disable IP forwarding
2. Limit services
3. Monitor log files carefully including logfiles on any services running.
4. Limit write access to files and directories on the firewall.
5. Implement policies to prevent denial of services attacks along with IP spoofing and IP fragmentation attacks. Enabling user quotas can help prevent denial of service attacks.
6. Limit access to services with the hosts.allow and hosts.deny files.
7. Set parameters in your TCP wrapper and any other services to protect against anyone pretending to have another host's name or address. See the section on inetd.
8. Be sure your /etc/securetty file will not allow root logins from unsecured locations.

System monitoring

Check your system log files often. They are in the /var/log directory. Check the log files /var/log/secure and var/log/messages daily. Also carefully monitor log files on any extra services you are running on your firewall.

General network policies

- Configure the identd protocol to allow for user name lookups from client to server machines. This will make it easier to track down any user who abuses the system.
- Use network monitoring tool software to detect abnormal activity on your system and intrusions.

The Chroot environment

A chroot environment is an isolated environment which is separate from the real operating system. It has its own root environment complete with necessary programs, libraries, and modules required to run independently of the real operating system. In this way it can become more difficult to break into the real operating system and damage it. A program that has root privileges, can, however still get into the real operating system but it becomes more work for an intruder. Some network services that can be configured to run in a chroot environment include bind and Apache. This chroot system is designed to prevent someone who has exploited a security problem in a service from getting access to the real system.

以上

問題2

下線部のみを和訳して提出してください。

Get on the D-BUS

Programs, the kernel and even your phone can keep you in touch and make the whole desktop work the way you want. Here’s how D-BUS works, and how applications are using it.

D-BUS is an interprocess communication (IPC) system, providing a simple yet powerful mechanism allowing applications to talk to one another, transfer information and request services. D-BUS was designed from scratch to fulfill the needs of a modern Linux system. The initial goal of D-BUS is to be a replacement for CORBA and DCOP, the remote object systems used in GNOME and KDE, respectively. Ideally, D-BUS can become a unified and agnostic IPC mechanism used by desktops, satisfying their needs and ushering in new features.

D-BUS, as a full-featured IPC and object system, has several intended uses. Firstly, D-BUS can perform basic application IPC, allowing one process to shuttle data to another-think UNIX domain sockets on steroids. Secondly, D-BUS can facilitate sending events , or signals, through the system, allowing different components in the system to communicate and ultimately to integrate better. For example, a Bluetooth daemon can send an incoming call signal that your music player can intercept , muting the volume until the call ends. Finally, D-BUS implements a remote object system, letting one application request services and invoke methods from a different object- think CORBA without the complications.

Why D-BUS Is Unique

D-BUS is unique from other IPC mechanisms in several ways. First, the basic unit of IPC in D-BUS is a message, not a byte stream. In this manner, D-BUS breaks up IPC into discrete messages, complete with headers (metadata) and a payload (the data). The message format is binary, typed, fully aligned and simple. It is an inherent part of the wire protocol. This approach contrasts with other IPC mechanisms where the lingua franca is a random stream of bytes, not a discrete message.

Second, D-BUS is bus-based. The simplest form of communication is process to process. D-BUS, however, provides a daemon, known as the message bus daemon, that routes messages between processes on a specific bus. In this fashion, a bus topology is formed, allowing processes to speak to one or more applications at the same time. Applications can send to or listen for various events on the bus.

A final unique feature is the creation of not one but two of these buses, the system bus and the session bus. The system bus is global, system-wide and runs at the system level. All users of the system can communicate over this bus with the proper permissions, allowing the concept of system-wide events. The session bus, however, is created during user login and runs at the user, or session, level. This bus is used solely by a particular user, in a particular login session, as an IPC and remote object system for the user's applications.

D-BUS Concepts

Messages are sent to objects. Objects are addressed using pathnames, such as /org/cups/printers/queue. Processes on the message bus are associated with objects and interfaces implemented on that object.

D-BUS supports multiple message types, such as signals, method calls, method returns and error messages. Signals are notifications that a specific event has occurred. They are simple, asynchronous, one-way heads-up messages. Method call messages allow an application to request the invocation of a method on a remote object. Method return messages provide the return value resulting from a method invocation. Error messages provide exceptions in response to a method invocation.

D-BUS is fully typed and type-safe. Both the header and payload of a message are fully typed. Valid types include byte, Boolean, 32-bit integer, 32-bit unsigned integer, 64-bit integer, 64-bit unsigned integer, double-precision floating point and string. A special array type allows for the grouping of types. A DICT type allows for dictionary-style key/value pairs.

以上