Florida State University
College of Arts and Sciences
Department of Computer Science

Creating a Security Laboratory Environment for Education, Research and Outreach

- ***Jennifer Frazier***

Project Submitted to the Department of Computer Science

in partial fulfillment of the requirements for the

Master of Science degree (M.S.)

Acknowledgments

Above all, I'd like to thank God who makes all things possible. I'd like to thank my parents who give me the never-ending support and encouragement through my project and always. Alec Yasinsac, for giving me guidance, pushing me to meet my goals and also helping Marion and me in publishing our first paper. I'd like to thank Steve Oberther for being there to answer my many questions. Also, I want to express my gratitude to my other committee members, Mike Burmester and Lois Hawkes for working with me in reviewing my project and attending my defense. I'd like to thank Wayne for hiring me for the systems group where I have learned so much that will help in my future as it has with this project and for finally letting me finish the DIS. Last but not least, I'd like to thank Ted Baker, who through repeated courses, has shown me how much better of a student I can be.

Table of Contents

1. Introduction

1.1 Motivation

In this current day and age of technology there is a rapidly increasing number of computer users and growth of the Internet. However, this has also led to an increasing amount of security risks arising from this growth. These risks have led toward the importance of education in computer security and the need for specialized computer security laboratories to aid in learning. The computer science department at Florida State University, through the funding of the FSU College of Arts and Sciences, has begun the steps toward turning a computer laboratory into the Security and Assurance in Information Technology (SAIT) laboratory. This security laboratory will serve three main purposes: education, research and outreach.

1.2 Education, Research and Outreach

The main focus of SAIT laboratory is to aid in the education of computer security. This is accomplished by providing students the opportunity to get hands on experience through lab assignments in correlation with classes. This allows the student to gain a better understanding of the material through implementation and practice in addition to learning through textbooks and lectures. Security classes instead of merely discussing how different protocols, securing practices, and encryption algorithms are used will now be able to elaborate these topics by installing, configuring and testing topic related programs in the laboratory.

Labs assignments, in combination with classes, will allow students to get more than a surface understanding of these topics, they will see the steps and execution methods involved in implementing and executing these tools and securing procedures. When it was decided to further develop the area of trusted systems, the computer science department hired three professors who specialize in this area. These professors created the criteria for classes geared toward computer and network security. This allowed the computer science department to award students who fulfilled the security class requirements the National Security Telecommunications and Information Systems Security (NSTISSC) certificate. This in turn brought about the recognition by the Nation Security Agency of the computer science department at Florida State University as a center for academic excellence in Information Assurance Education.

SAIT laboratory is a dedicated research center for students and faculty. Here they have an isolated environment, on its own local area network (LAN), which gives the freedom to perform tests and have the system maintained in response to the needs of the security laboratory. One of the benefits of being on its own network is that it allows the laboratory to test different security measures without affecting the regular network. This can include anything from running packet sniffers to port scanners. This could be a potential security threat due to the fact that packet capturing may be used to obtain passwords. Scanning tools may inform of potential system holes that may be used to exploit the main system. Being on separate network alleviates this problem.

Another benefit to using a separate network for SAIT is that it allows students to perform testing which may tie up the network and, if on the computer science network, cause a denial of service attack that would keep legitimate users from gaining access to the systems. Also, this allows testing to be done in a controlled environment. This includes having the computers configured to allow users to install programs on the machines and being able to generate specific traffic. This also includes the ability to block outside influences which could alter testing results such as network traffic or user load on the system.

The SAIT laboratory Outreach program fosters communication and endorses mutually beneficial relationships among members of government, industry and the academic community. These affiliations with sponsors (such as industry and government) through token investments, gain access to the laboratory and partner access to the professors and members of SAIT. The laboratory in return will be provided with access to current software resources and equipment to perform cutting edge security education and research.

1.3 SAIT Laboratory

Due to the capabilities required by the SAIT laboratory, a variety of equipment was purchased towards meeting those needs. The laboratory computers operate on a separate network in order to partition them from the computer science network. This is primarily for security purposes, and additionally protects the computer science department from denial of service attacks and other threats from the laboratory operations.

Different architectures and operating systems were obtained to allow a broader spectrum of learning and research.

These items include:

- 1 Sun Microsystems Enterprise 220R Server
- 1 Cisco Catalyst 3500 Series XL Switch
- 1 Sun Microsystems StorEdge A1000
- 9 Sun Microsystems Ultra 5 Workstations running Solaris 8 and equipped with a PCI card capable of running the Windows 2000 Professional operating system
- 5 Custom Build PCs with Windows 2000 Professional
- 4 KVM Switches
- 1 HP LaserJet 4100N Printer

A setup diagram of SAIT laboratory is included in Appendix [C].

Using this equipment different simulations can be created in SAIT. One setup may be used in creating a honeypot to attract potential intruders to analyze and research their actions. Another setup could aid in researching different attacks methods performed on two communicating computers such as the man in the middle attack.

2. Project Statement

2.1 Objective

There are two different objectives to this project. The first deals with creating a UNIX environment on the Ultra 5 Workstations for users. The second objective involves creating a database of UNIX security tools.

Creating a UNIX environment arises from two different needs. The first problem a user encounters when using a UNIX machine in SAIT laboratory arises when trying to install software. If a user is given a class assignment to carry out in the laboratory which involves installing, configuring and testing a program, there are many problems they will encounter with this simple classroom assignment:

1. Computers in labs are designed such that regular users may not install programs onto them and may only use the programs that already reside on the system. Only system administrators may install software on the machines.

2. Programs installed could maliciously interact with other programs or unnecessarily consume large amount of hard drive space.

3. Another problem can occur if users are able to delete items from the machine. Should programs or integral parts of the file system happen to get altered or removed, this can result in the computer or program functioning incorrectly or not at all. Situations that arise like this are hard to assess and remedy due to the fact it is difficult to determine everything that has previously been done to the computer.

All these factors create a need to give the user an environment which will grant the flexibility and privileges needed to install and configure software, yet be constraining enough so users will not be able to cause harm to the operating system and any other important information or programs which resides on the system. Also, once the user has finished their session on the machine, the environment should go back to an initial (default) state as if the user had never sat down at the computer. This is accomplished by re-initializing the environment and gives each user a fresh untouched working environment when seated at the computer.

The second part of this project involves creating a database of UNIX security tools. This requires deciding upon categories for tool placement followed by researching, installing and execution of each tool collected. These categories include: Firewalls, Intrusion Detection, Vulnerability Management, Authentication and Encryption, and Miscellaneous.

2.2 Scope

The scope of this project is to develop the UNIX environment in SAIT laboratory which will aid in its mission to promote education, research and outreach. This will be accomplished by:

1. Find a way to perform security studies on the Ultra 5 Workstations.
    - Researching and collecting a database of enough security tools to be representative for each of the proposed security tool categories. The different categories of security tools are discussed in section 3.3.
    - Granting users enough permissions to be able to install and execute security tools
2. Keeping the overall system protected from the users.
    - Creating a sandbox user environment which contains a miniature filesystem that only utilizes necessary files and commands to allow users to install, configure and test security tools, yet keeps the system safe from users. Once the user has logged out of the system, the machine will be restored back to an initial (default) state.
    - Find a method to update client environments from the server.

3. Methodologies

3.1 Different Environment

There were many different methods to consider when choosing the user environment to use for SAIT laboratory. The different ideas along with their pros and cons are discussed below.

3.1.1 Cloning

The first method considered to create the user environment was cloning the hard drive each time a user completed a computer session to bring it back to its default settings. Cloning creates an exact image (or snapshot) of a hard drive that may be used to install new or revert other hard drives to an original desired state. A common program used is Ghost that has coined the term ghosting to mean cloning hard drives. However, this idea had multiple drawbacks. The first problem is that cloning does not create a solution for insufficient privileges when trying to install security tools.

Another problem with cloning deals with hard drive wear and tear. Current ghosting technology allows partitions that the software recognizes to be selected and re-imaged. Although this saves on the amount of hard drive space which needs to be ghosted, this still creates an unnecessarily large amount of wear and tear in removing and recreating large portions of the hard drive. This repeated disk access can lower the life span of the drive by decreasing the mean time between failures. Also, the chance that the whole environment would constantly need to be recreated is unlikely and requires large amounts of time to clone the drives. Due to this loss of time, having to ghost drives during regular laboratory hours results in sacrificing laboratory usage time.

3.1.2 Non-Proprietary

The next idea involves using software (proprietary or non-proprietary) to create the necessary environment. First examined were different non-proprietary softwares. This includes programs such as User Mode Linux [51] and Chroot Jail [9].

User Mode Linux has the capability to allow users to have the necessary permissions to install software and to bring the machine back to a default state when the user completes their session. This program functions similarly to the way the Windows environment for users was created in SAIT, using VMWare [8]. The main idea behind VMWare is that it places a “guest” operating system on top of the actual “host “operating system, which is the basically the same method used in User Mode Linux. However, although it can run on the Sparc architecture, it is only available for Linux operating systems.

On the other hand, Chroot Jail had a great basis for a program to be used for the SAIT environment. Chroot Jail runs on the Solaris operating system, but does not contain all the necessary functions required to create the desired environment. However, this program lays a great foundation for the idea behind J’s Secure Sandbox (JSS). This is discussed more in section 3.2 and additionally the shortcomings of Chroot Jail which JSS adds or improves upon.

Although both these programs proved to be good possible solutions, they either did not work with the Solaris operating system, the Sparc architecture or fulfill all the needed requirements. Further research concluded there was no available freeware that could create the desired user environment. This led to the search for proprietary software to generate the controlled environment.

3.1.3. Proprietary Software

The first proprietary program researched was Deep Freeze. This software had the capability to "freeze" a configuration file that is used to create the system. It also grants users necessary permissions to make changes and install programs to the machine while the program is in the "freeze" mode. This way, although it appears to the users as if their changes are taking place, once

the computer "unfreezes", the environment would go back to its original state. Unfortunately this software only supports the Windows environment.

Another proprietary program considered was VMWare. This allows operating systems (aka guest operating system) to be installed on top of the actual operating system (aka host operating system). Once the guest operating system has been installed and the user has started the VMWare program, the guest operating system is booted. This program allows users to have super user privileges on the guest operating system, while not allowing changes to take place on the host. When the user closes VMWare, it shuts down and removes any changes made to the guest operating system by the user. Although this program works with many different operating systems, it does not work with the Solaris operating system or the Sparc architecture and is not guaranteed to work on Solaris for the Intel architecture.

Installing the Solaris operating system for the Intel architecture onto the PCI cards was also attempted to try and provide the requirements for VMWare. The logical reasoning behind this endeavor stemmed from the fact that PCI cards have the necessary Intel architecture and run the Windows 2000 operating system. However, the VMWare program crashes when trying to install the Solaris for Intel operating system.

The final working solution was to write software to create the necessary working user environment which would grant users permissions to install security tools while maintaining a safe environment. Thus was born JSS.

3.2 Related Work

The solution used on the Windows side of this project involves using a wrapper to create the necessary environment for the user. The proprietary program VMWare creates a layer between the host operating system and the guest operating system so that any changes made on the guest is not able to affect the host. Wrappers may be defined as “a program that sets up another program so that it can run successfully” [53]. However, JSS uses the idea of a sandbox to create its environment. “Sandbox restrictions provide strict limitations on what system resources users can request or access” [42]. This is known as the area where users may play freely but only within the confinements of the sandbox and nowhere else. This sandbox is the safety logic behind JSS that keeps users from causing any harm or changes to the actual filesystem.

Inside the sandbox users obtain the necessary permissions to install and execute software. This miniature replica of the filesystem only includes necessary commands and files for the environment. The user is now able to seemingly get into parts of the filesystem which they previously could not access. Being denied into certain directories and files is the main inhibitor for users when trying to install programs. Inside this sandbox, users will have temporary ownership (while executing this program) of JSS, and during their session they seemingly control this filesystem.

3.2.2 Chroot Jail

The idea for JSS came from a similar program entitled Chroot Jail [9]. Chroot Jail is an open source freeware program. The motivation behind Chroot Jail is to build a secure environment as portably and simply as possible. The program gets its name from the command that carries out the main function of the program, chroot. The man page for chroot states "The chroot() and fchroot() functions cause a directory to become the root directory, the starting point for path

searches for path names beginning with / (slash). The user's working directory is unaffected by the chroot() and fchroot() functions" [13]. Simply stated, this allows a specified directory to become the new root directory for the process. To invoke chroot the effective id of the calling process must be root.

Chroot Jail recreates a basic part of the file system inside the chroot jail and copies over an implemented set and also specific user requested commands.

3.2.3 Improvements

Although Chroot Jail is highly portable, it lacks certain essential requirements for the SAIT UNIX environment.

The first problem with Chroot Jail is the lack of programs, libraries and directories that are copied into the user environment to install security programs. Although this setting allows the user the necessary permissions to install software, the components that Chroot Jail supplies will only support enough functionality to create the user shell and the most basic commands. On the other hand, if the user knew every library, command and program associated with the software involved, invoking the correct commands when creating Chroot Jail, they would successfully be able to install that single tool into the environment. Also, Chroot Jail does not port a browser into its secure environment.

The idea in creating the SAIT UNIX environment is to have a fully prepared environment for a user to be able to sit at a computer and install security tools. To make this program usable for our purposes, an improvement to Chroot Jail would be to create the environment already containing the programs, directories and libraries to install security tools from the collected database.

Another functionality not included in Chroot Jail is the ability to use the program on multiple machines and maintain the same working environment on each of the clients. Chroot Jail was created to work on stand-alone machines. A way to implement a solution to this problem is to have a default sandbox reside on a server and then to have each of the client machines update their environments from the server which would result in the clients having exact replicas for their user settings.

For a user to have a fresh copy of the Chroot Jail filesystem, the entire sandbox would need to be removed and then recreated which could cause unnecessary wear and tear to the hard drive depending on the size of the Jail. To get around this drawback JSS only recreates parts of the default environment which have been removed or changed, and delete anything on the client machine which does not reside on the default operating system.

Finally, there are certain parts of the filesystem necessary for the sandbox environment for our purposes that the creator of Chroot Jail did not implement. In order to work with different operating systems and architectures the program must be kept as simple as possible. With the basic concept of Chroot Jail in addition to these mentioned improvements, this was the layout to create a sandbox for the SAIT user environment.

3.3 Overview of J's Secure Sandbox

JSS is an implementation of the sandbox concept discussed in section 3.2. JSS creates a condensed filesystem inside of the actual file system. While the user is logically inside the secondary file system and inside this sandbox, they will have ownership of all the files on what

looks like the actual file system, but will not actually be able to access or damage the actual file system. A user will go through the initial logon process into a computer with their username and password for their account on the computer's system. After the user has successfully logged onto the computer, they begin the JSS program.

The program initially creates a secondary file system to be used as the sandbox. This UNIX file system contains only the minimum necessary directories, programs and libraries for basic functionality. A unique user and group ID (jss_user) is assigned to the sandbox file system and temporarily, while the program is running, the user.

Once inside the sandbox, the user is not able to access any part of the actual file system until they log out of the program. This secondary file system appears to the user as the actual file system but they have the flexibility and virtual super user privileges to install software without affecting any part of the main file system on the computer due to being confined in the sandbox.

3.4 Security Tools

3.4.1 Categories

Before the search could begin for security tools, there had to be a decision made on the different categories of tools and their embodied characteristics. Due to computer security being still a fairly new topic, no standard guidelines have been created for the different security tool categories. Based on all the information gathered, Table 1 lists the security categories and criteria for the database of tools.

Firewalls	
Any set of related programs that examines packets and decides whether to forward it towards its destination.	
Intrusion Detection Systems	
Real time monitor system that compares ongoing activity in the target domain to signatures or profiles to detect malicious or abnormal activity.	
Network Based	Intrusion detection program that monitors an entire network.
Host Based	Intrusion detection program that monitors a single host.
Sniffers	Packet Capturing Program.
Auditing Tools	Constant system monitor for sundry services (log files, resources).
Hybrid Tools	Intrusion detection program that has functionalities of host and network intrusion detection system such as monitoring an entire network or single computer.
Vulnerability Management Tools	
Tools which help to reduce sources of known vulnerabilities on machines.	

Lockdown	Defines access controls for resources on a machine to help reduce vulnerability on a machine.
Password Assessment	Checks for weak (easy to guess) passwords.
Port Scanners	Scans the ports of a machine to see what services are running.
Security Assessment	Scans a machine or multiple machines to detect known vulnerabilities.
Authentication & Encryption	
Tools used for the purpose for the authentication of users which goes hand in hand with encrypting data for security purposes.	
Encryption/ Decryption	Programs which use algorithms that will encrypt and decrypt data.
Hashes	Programs that use one-way encryption algorithms.
GPG	Freeware for PGP - Public Key Encryption.
PGP	Public Key Encryption
Miscellaneous	
Testing	These are tools which test or aid in the accuracy testing of security tools.
Front End	Tools used in addition to other tools. They are normally GUI interfaces or output parsers.
Media Cleanser	Tool which aims toward the complete removal of data from a portion of the hard drive.

Table 1: Security Tool Categories

Appendix [C] gives a complete listing and read me of the researched security tools.

3.4.2 Software Types

Once the security categories had been established, the research of tools began. There are three types of software; Proprietary, freeware (which includes open source) and shareware.

Proprietary software must be purchased. There are many benefits to paying for software: users receive the full functionality of the product, receive product updates and bug fixes, and may obtain user support.

Shareware normally asks the user for a small donation for usage of their product. This can be anything from requesting e-mail addresses to sending the program writer a token sum. However, shareware is normally a beta version of the software and does not provide full functionality.

Freeware tools are free for anyone to use and do not require anything in return. Freeware can either be closed or open source. Closed source programs do not give the user access to any of

the underlying code while open source makes it sources open to the public. All the UNIX security tools researched for this product are open source freeware tools.

There are related difficulties when using open source freeware. Firstly, there is no guarantee that the program will be able to execute correctly on your system. In these cases there is usually little or no user support available. Also, the contents of the program (unless the code is carefully examined) could contain something malicious. Although this is normally not the case with freeware, it is a situation that can occur and therefore shows the importance of obtaining security programs from trusted companies and web sites.

Many times open source freeware tools require are not standalone, but require installation of additional programs and libraries to execute correctly.

Benefits to freeware include that these programs may release updates more frequently and be more responsive to releasing bug fixes for new arising security holes while proprietary software vendors may only release updates to their software at set intervals throughout the year. Also, when user support is available for a freeware product, direct contact with the original software creator or maintainer is normally possible.

Although source code adjustments are usually necessary to get programs to execute in an environment, this is also one of the best features of open source freeware. This allows users to alter code to fit their specific needs and to fix arising problems running the software on their system. Also, freeware eliminates concern dealing with copy write and licensing issues such as when using their product for SAIT laboratory.

4. Requirement Analysis

4.1 Design Problems

One of the first design problems involves granting the user enough permission to install software yet restrain them from being able to cause any harm to the machine. The main restrictions on program installation deal with users not having access to certain directories. Once the program has been installed into the directory, root permissions are normally not even required to execute the programs. Security programs often times recommend not running the program as root. Keeping the main filesystem secure is the main priority when allowing users to install and execute software. The way to restrain users from causing harm to the machine is to simply remove possible ways to access the filesystem. In JSS, this is accomplished by calling chroot in the program's user directory, jss_user, which makes it appear to the user as if they are the super user on the machine, but are actually safely contained inside a sandbox.

Another design problem is working with the Sparc Ultra V's which only contain 8 gigabytes (GB) of hard drive space. This is currently not a problem for this project because the sandbox is only about 300 megabytes (MB), but additional hardware should be purchased for future uses.

Determining when to update the sandbox environment was another design consideration. Reasons to update the sandbox upon program invocation include:

1. If software has been updated on the server it will immediately be available to the clients.
2. If the client environment is updated directly before the user is launched into the sandbox, they receive the most recent copy of the sandbox. This would possibly not be the case if updating were to be conducted following user sessions due to the fact that there could be a lengthy period of time before another user executes the program. In

this lengthy period, many updates could have been done on the server and the sessions would not reflect these updates until after the user completed their session.

Both methods of updating, before and after program execution, aid in maintaining a constant working environment across the clients. Clients are able to update their own environments by connecting to the server instead of each client needing to be separately updated by a system administrator.

Other security considerations include:

1. Users trying to abuse their upgraded privileges
    a. Keeping the mind that the main priority is securing the actual filesystem from users, the main safety feature of JSS is due to the fact that at no time does the user ever receive root permissions. During the entire execution of the program, the only permissions that the user receives are those of the specially created program user, jss_user. Also, the user does not have access to anything that would allow them to escape from the sandbox nor affect any part of the actual filesystem.
2. Connecting safely from the client to the server when updating the environment
    a. The program rsync is used for connection between the client and the server. It has been set up so that an rsync daemon is running on the server which will only accept connections from specific users and, once authenticated, only allows them to update from a certain path (directory) on the server. The client is set up to enter the necessary username and password to gain access to the server while users are only aware that updates are taking place.
3. Programs for JSS need to be checked to ensure they do not contain any major security holes.
    a. The only program required of JSS is rsync. The latest version is 2.5.5 (April 2, 2002) with no known vulnerabilities. This version includes a fix for the recent zlib double-free bug.

4.2 Design Decisions

The first decision was selecting which languages should be used to write the scripts and main program for JSS. Perl was the first choice to use to write the sandbox creation script. However BASH was chosen because it comes standard on many UNIX systems. C++ was used to write the main program due to the fact it is the most familiar language to the creator.

The next decision involved whether to recreate the entire filesystem inside the sandbox or only select segments of the filesystem. For this project only portions of the filesystem were copied into JSS. Flat files direct the specific commands, directories and libraries that are brought into the sandbox. These flat files are read into scripts which seek the sources on the actual filesystem and then copy them into the JSS environment. This method also takes into account space constraints of the machine (hard drive space) and time constraints of building the environment.

A consideration when creating JSS was the methods used in creating and copying the environment on the machines. The methods are discussed above in section 3.1. It was decided that creating the software and using the rsync utility to copy the sandbox from the server to the clients was the best decision. Rsync security features are discussed in the above section 4.1.

Another decision involved whether to remove and recreate the entire sandbox or to only restore altered portions of the sandbox to update the environment. For the original default sandbox on the server, it is best to recreate the sandbox each time the sandbox generation script is run. The reason being that the script only needs to be rerun when updates have taken place on the server and recreating the filesystem ensures to bring all the updates into JSS. The clients only need to restore the altered portions of their filesystem. Due to the more constant updating of the environment (each time a user executes JSS) this saves unnecessary wear and tear on the hard drive.

5. Designing J's Secure Sandbox

5.1 Layout

A script which resides on the server (startup.sh) will create the default sandbox environment. If any updates take place on the server after the JSS environment has been created, the script only needs to be rerun to update the environment to the most current settings. Updates will normally include any type of software upgrade made to a program or library on the server which is used in the JSS environment.

The main program executable will be placed on each of the client machines. Each of the sandboxes created on client machines will rsync with the server to ensure they have an exact replica sandbox to maintain the same environment on all the clients. The client program will update in the beginning in the execution of the program. This ensures if any updates/changes were made on the server, they will be reflected on the clients before the user is launched into the program environment.

Once the client program has created the environment and set the permissions, it will launch a bash shell for the user to work in. This shell will allow users to install the database of security tools. Once the user types the word "exit" to close their JSS session, the program closes the user shell, resets any local changes made to the host file-system and exits. The security tools will be a part of the default JSS environment, each individually compressed in a tool directory along with any programs or libraries necessary for the security tools.

5.2 Setting up the Server

The first part of creating JSS was to establish files containing all the necessary commands, libraries, files and directories. These will be used to create an untouched sandbox on the server that all the clients will use as their base model.

The main script (startup.sh) begins by checking to see if the destination directory (where the original default sandbox is to be created) exists. If the directory exists, it deletes it and then recreates it. This will update the sandbox in case any changes have taken place on the server since the last sandbox creation. The startup script contains five different programs which each perform a different operation for the filesystem. To update certain portions of the filesystem, only the corresponding script needs to be executed instead of the startup script which recreates the whole environment from scratch. This also allows for easier debugging and editing of the scripts.

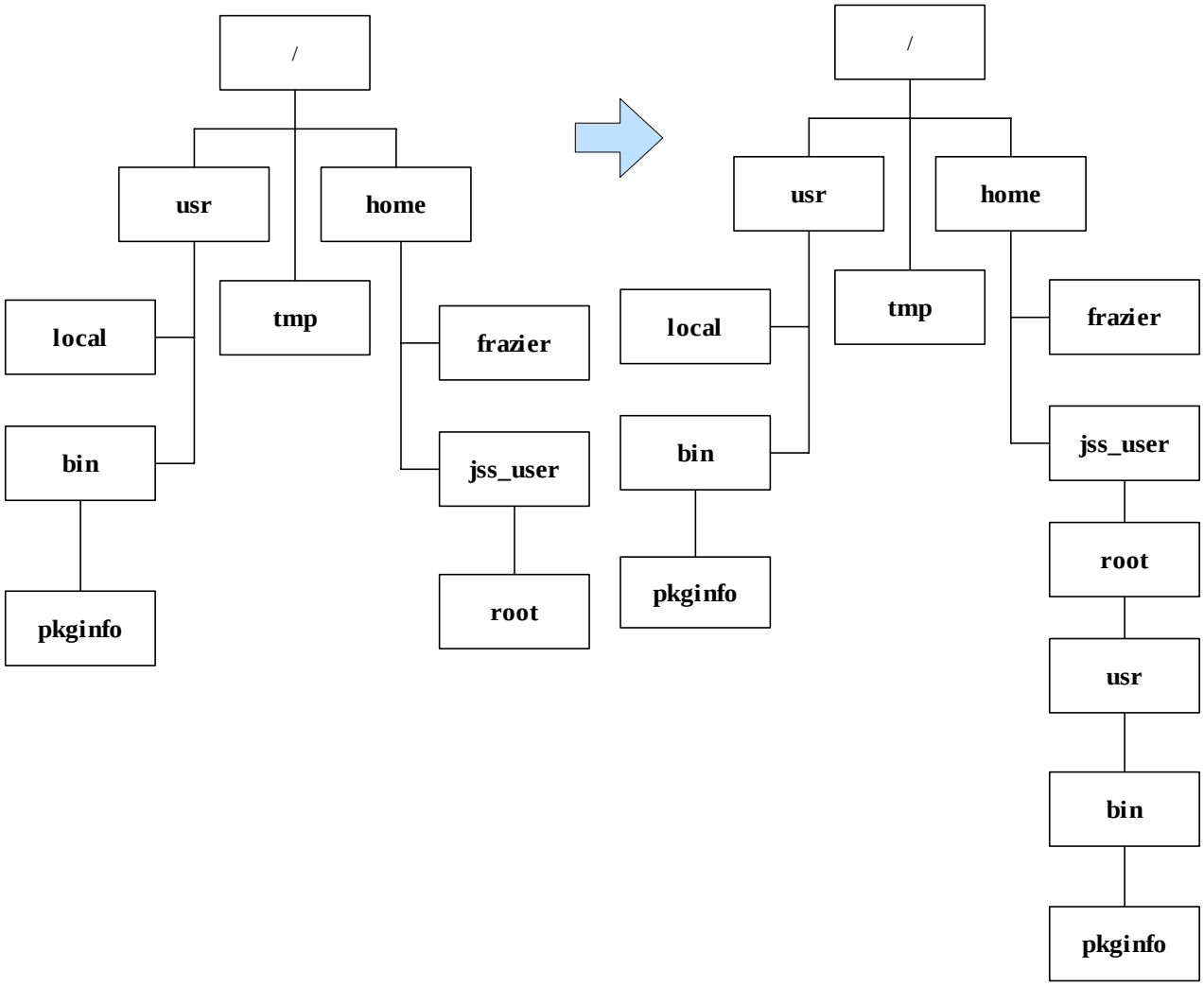
The first script (find_lib.sh) checks for its associating commands file list (commands_list). If the file is not found, the program exits. For each of the commands listed in the file, the program gathers its path information and all the libraries associated with the command.

When a command is found it outputs the command name and path to the screen, else it outputs an error message claiming the command could not be located and would not be installed

into the sandbox. The program then proceeds to copy the command and its full path into the sandbox.

Figure 1 shows an example of how commands are copied into the sandbox maintaining their original structure of the directory tree. In Figure 1(a), the program “pkginfo” is currently being copied into the sandbox. The full path for “pkginfo” is “/usr/bin/pkginfo”. However, the parent directories “usr” and “bin” have not yet been created. Figure 1(b) shows how the program will create and copy these directories maintaining their full path structures into the filesystem, and then copy the “pkginfo” command. “/usr/bin/pkginfo” has now been copied into “/home/jss_user/root/usr/bin/pkginfo”.

The pkginfo command is copied to /home/jss_user/root/usr/bin/pkginfo, the parent directories /usr/local have not yet been created inside the sandbox and will also be copied maintaining the same file structure.



(a) **(b)**

Figure 1: Copying Directory Structure and Command

This script then checks to see if the corresponding output library file exists (lib_file). If the file exists, it is deleted and then recreated. This way only the libraries associated with the latest commands are copied into the JSS filesystem and no unnecessary libraries. Once the script finishes finding the associated libraries, it performs a "sort" which alphabetically sorts the libraries and executes "uniq" which removes all duplicate libraries from the file. The libraries are then copied into JSS.

The directory and file specific script (find_dir.sh) operates in a similar manner to the command script. It checks to see if the associating file list (dir_file_list) exists. If the file does not exist, the program exits. It proceeds to go through dir_file_list and copies the directories and files into the JSS filesystem while displaying on the screen what is currently being copied and whether it is a file or a directory. At the end of this script it creates a "var/tmp" directory in JSS due to the fact that this directory does not need to be copied from into the JSS filesystem, but only created so that when installing programs, they may place temporary files into that directory.

When copying .bashrc, a .bashrc directory was being created using the file name and placing the file inside that directory. The new algorithm solves this by removing the additional dot which was causing the problem.

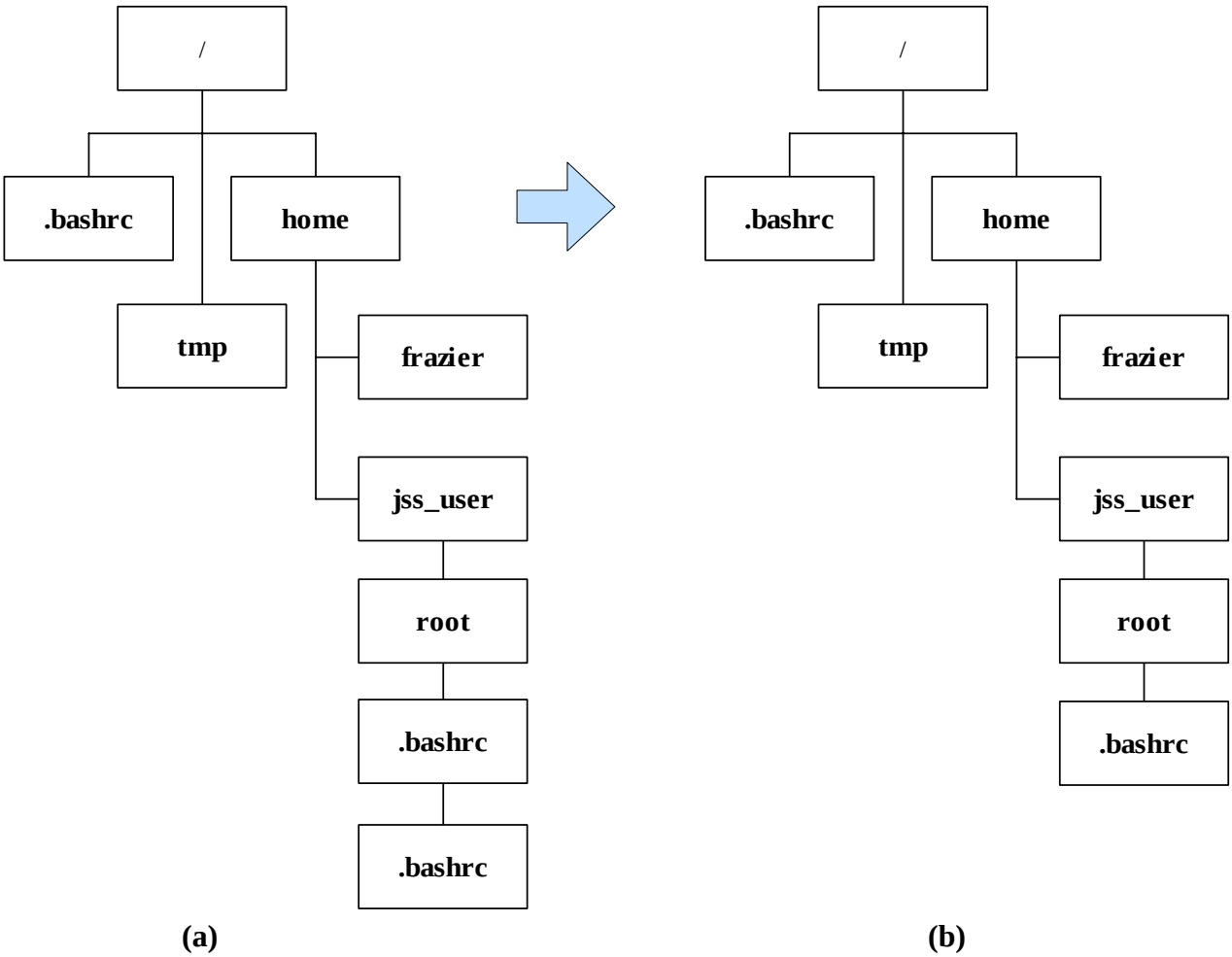


Figure 2: Copying Dot Files

The next script (find_dot.sh) specifically copies over dot files and compressed (tar and gz) files. The previous scripts copied these files over as directories and then placed the file inside the newly created directory. Figure 2 shows this problem with the dot file ".bashrc". Figure 2(a) shows how the scripts mentioned previously incorrectly copied the file. The correct filesystem layout is shown in Figure 2(b). To solve this problem the find_dot.sh script was created.

The next program (mk_syms.sh) replicates symlinks in the actual filesystem into the JSS environment. Due to the fact that some programs look for specific path names for libraries and files, such as /lib which is actually a symlink to /usr/lib, this creates the need to replicate these symlinks in the JSS environment so that installation programs will not know the difference between JSS and the actual files system.

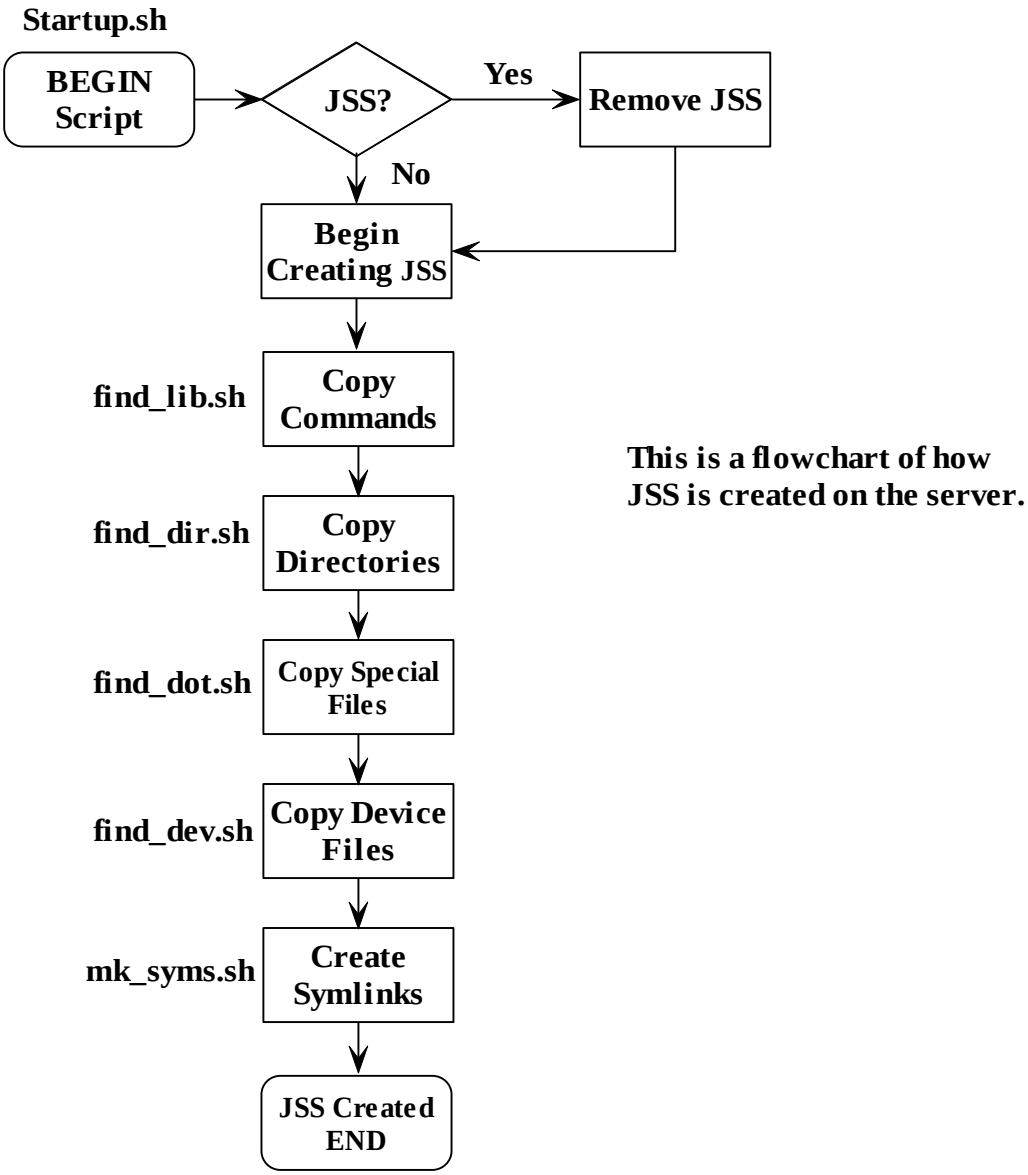


Figure 3: JSS Server Creation

The final script copies the system device files (find_dev.sh). This was accomplished using the "mknod" command which creates special files for character and block devices. Character special files are used for unbuffered data transfers to and from a device. This includes any device that accesses a single character at a time such as a terminal. Block files are used when data is transferred in fixed sizes known as blocks. An example of a device that transfers data blocks is the hard drive. Both kinds of files exist for some devices such as disks.

Figure 3 shows the execution process of creating the default JSS filesystem on the server. Appendix [D] contains the user and maintenance documentation for the JSS program. Appendix [E] contains the source code and related files to JSS.

5.3 Setting up the Clients

The client program is written in C++. This program executable is installed on each of the clients in the SAIT laboratory with the set user ID (SUID) bit enabled as root. This will allow the program to be executed as root, but will in no way allow the user to have any root permissions. The program begins by rsyncing the client JSS environment with the default environment on the SAIT server. This allows the user to have a fresh copy of the environment and additionally to have the most recent version of the sandbox.

The first time this program is executed on the client, it will take approximately five minutes to create the sandbox due to having to create the initial environment from scratch on the machine. This is the longest period of time that the program will take to create the environment. After the initial install, the only updates made to JSS will be those of recreating default settings that may have been removed and deleting items the user placed into the sandbox. After the rsync has completed, the program (changes the ownership) of everything in JSS to have the UID (user id) and GID (group id) of the program user, jss_user.

Next, the program mounts /proc in /home/jss_user/proc. Filesystems are attached to the directory structure with the mount command, and detached with the umount command. Due to the fact that nothing outside the environment will exist for the user once the chroot command takes place, proc will not work unless mounted inside JSS. Proc is a virtual file system that provides the capability to inspect processes.

Next, the program performs the chroot command. This command changes the root directory to a different specified directory and will create the sandbox environment to contain the user. The directory /home/jss_user/root will appear to the user as the / directory of the filesystem. After changing the real and effective user ID (UID) and group ID (GID) of the program to be jss_user, the program launches a bash shell which places the user inside the sandbox. When the user types "exit", the program regains real and effective ID and GID as root and then unmounts the proc filesystem from /home/jss_user/root/proc. Even though the user has terminated the program, the sandbox containing the user changes still resides on the system. The JSS filesystem will remain this way until the next user executes the program and the environment is updated from the server through rsync.

Figure 4 shows the execution process for the client program. Appendix [E] contains the user and maintenance document for the JSS program. Appendix [G] contains the source code and related files to JSS.

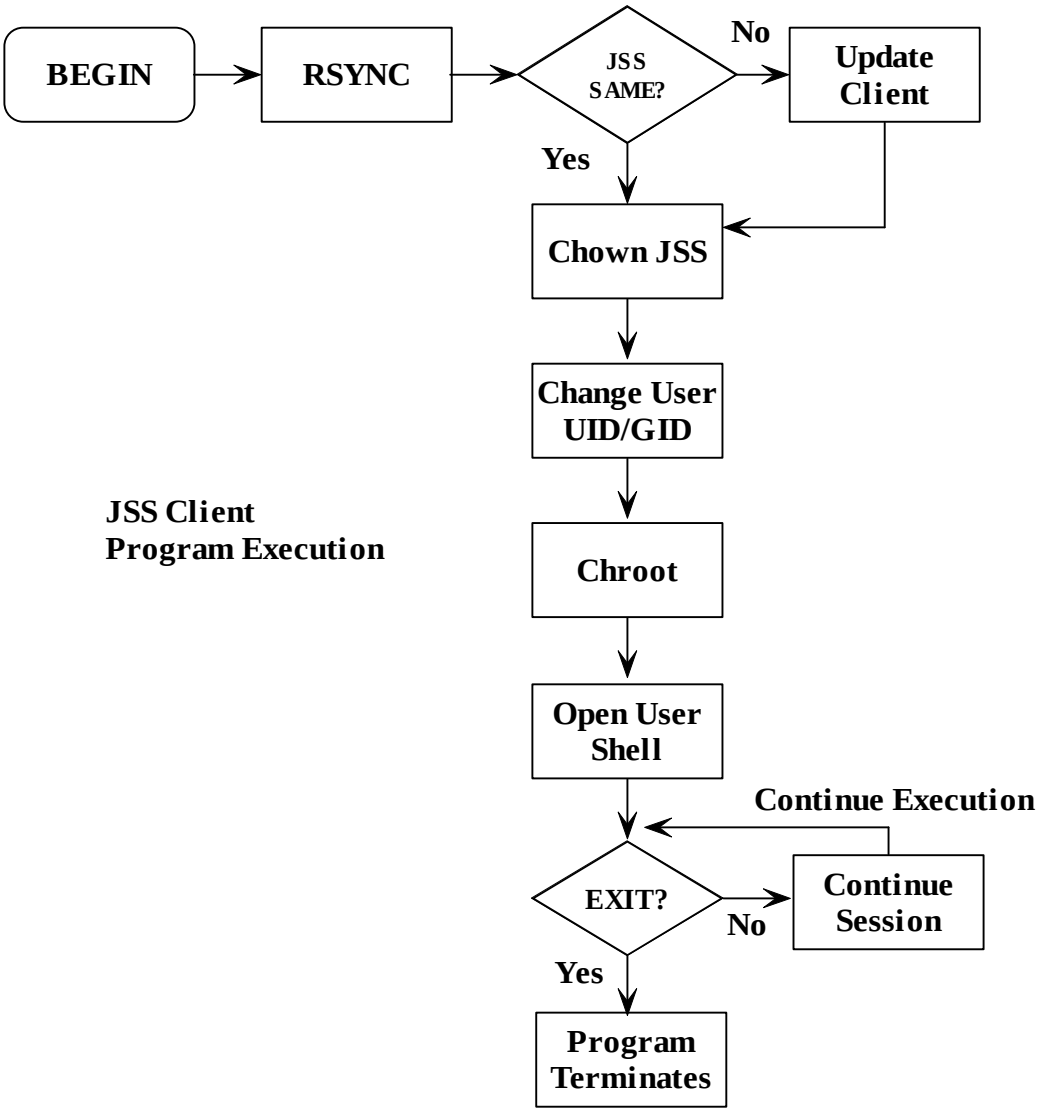


Figure 4: JSS Client Program Execution

5.4. Authentication between Client and Server

Rsync is an open source freeware program that is used to update the clients from the server [41]. It is an open source freeware program developed by Andrew Tridgell for variants of the UNIX operating system. Rsync is a replacement for the rcp command and includes many additional features. The rcp command copies files to and from hosts without supplying a password. The rsync README file states that rsync operates by "using the 'rsync algorithm' which provides a very fast method for bringing remove [sic] files into sync. It does this by sending just the differences in the files across the link, without requiring that both sets of files are present at one of the ends of the link beforehand."

Rsync allows the client machine to update its JSS filesystem with the original one located on the server through the use of the rsync algorithm. The algorithm begins by splitting files into fixed block sizes. For each created block, it creates two checksums, a 32-bit checksum (aka weak "rolling" checksum) and a 128-bit checksum. The client will send both of these checksums to the

server. The server uses a special property of the rolling checksum quickly in a single pass to find all the block sizes from the client. The program actually uses a 3 level searching scheme that is described in Appendix [F]. The server then notifies the client for only those pieces not found on the server. This method alleviates the need for copying complete files do to its manner of checking fixed block sizes which only requires copying altered portions of files.

Rsync is used for authentication between the clients and the server. On the server side, the file rsyncd.conf may be used for different access and user configurations for the rsync daemon. This file on SAIT controls the users which may access Rsync through the daemon and also which directories those users may access. The users are authenticated using a file rsyncd.secrets which contains the username and password pairs.

The client first connects to the server through the rsync daemon. The option --password-file allows connection to the server by using a file which stores jss_user's password. This way the user does not needs to enter or even know the password for the client to connect to the server when updating JSS. This is also a security measure which makes it unnecessary for users to have an account or password to an account for the server. Other options used in this program with rsync include links, verbose, checksum, recursive, delete, delete-excluded, progress, perms and force. The details of these options are included in appendix [F]. Once the client has finished updating, the connection between the server and client is closed.

Figure 5 shows the authentication and updating process between the clients and server. Appendix [G] contains the documentation and relating files for rsync.

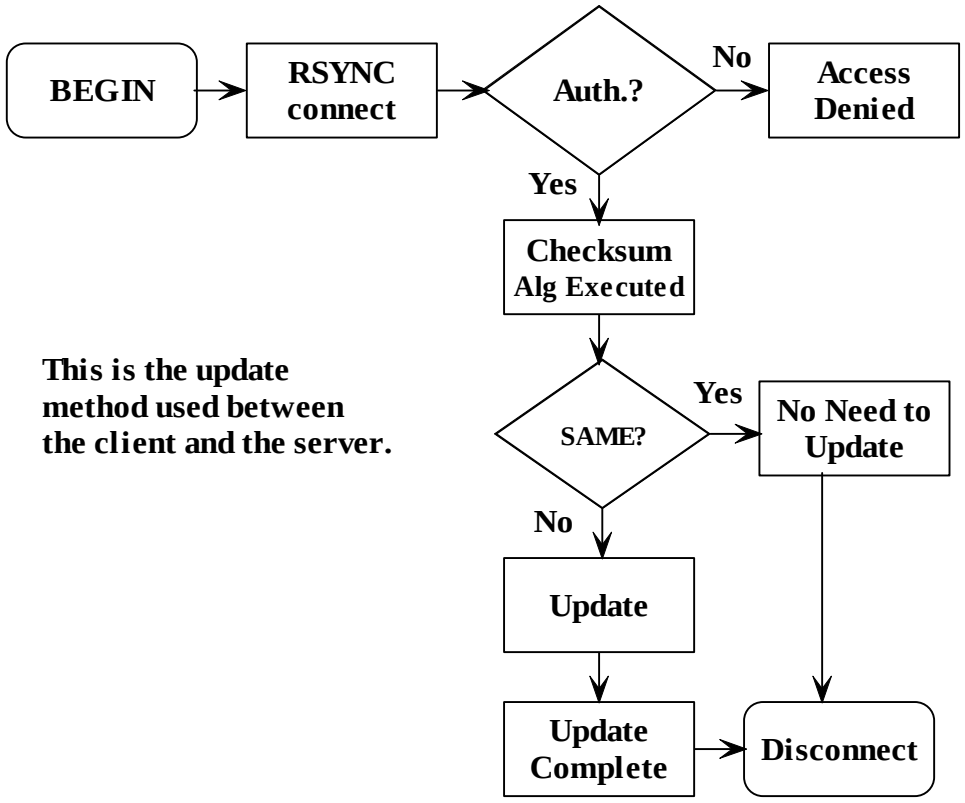


Figure 5: Rsync Procedure Between the Client and Server

6. Testing and Results

6.1 Test Design

Testing for this project was conducted by trying to execute different operations inside the environment to ensure they worked. This includes performing different simple commands such as ls, to using the vi text editor and opening the Netscape browser. Most importantly, ensuring the tested tools installed and executed inside the sandbox.

When trying to install the different tools or commands, if the operation could not be executed, one of the methods used to try and discover the problem was using the truss command. The truss command showed the system calls executed when trying to run a command. Other methods included studying output errors or trying to find solutions on the web. However, due to the fact that all these programs had been installed on the Solaris machine in the regular environment, it had to be a missing command or file causing the programs to function incorrectly in the environment. Other testing was done to ensure that all the necessary options were being utilized correctly for the rsync command to rebuild the environment. This included recreating files that had been deleted from the original environment or removing any additional items added by the user. Testing was also conducted to ensure the client could connect to the server without the user needing to enter a password.

Other testing included ensuring that changes inside the sandbox did reach and could not affect any part of the actual filesystem. This is aid in guaranteeing that the main filesystem remains unaffected by the users.

6.2 Validation Testing

These are the steps involved in testing the functionality of the JSS program, environment and security tools.

- The initial script (startup.sh) is placed and executed on the server to ensure proper creation of the original default filesystem.
- The program executable is placed on the client with the SUID root. This allows the program to run as root.
- The program executable is started which begins the rsync operation to ensure that the sandbox on the client is an exact replica of the one of the server.
- Tools were installed, compiled and executed on the clients to ensure they function properly.
- Once the user closes their JSS session and has completely exited the program, JSS is restarted which once again executes the rsync command, to verify that environment is being recreated. This ensures that the user environment is indeed being rebuilt on the client (gone back to the default version).
- Tools are tested inside the environment to ensure it contains necessary files and commands to install and execute the collected security programs.
- Execution of the program and then trying to start up following sessions to test the lock file of the JSS program.
- Trying to log into the server using different usernames and passwords to ensure the daemon is executing correctly tests the authentication process.

6.3 Performance Analysis

This is information about and execution times related to JSS. The times below are timed using the ptime command. The first result is "the elapsed (real) time between invocation of utility and its termination" which displays the time in hours, minutes, seconds and microseconds. The second result is the "user CPU time" which is the amount of CPU time spent executing user code. The final result is the "system CPU time" is the amount of CPU time spent executing system code. Both the user and sys CPU times give their measurements in clock ticks[49]. There are one hundred clock ticks in a second on the machine tested. However, the user and sys CPU times are not of importance here, but are included for user interest. All these test results are affected by traffic on the machines and network.

(a) Size of the JSS sandbox:

This measures the size of the original default sandbox that is created on the server and copied to all the clients. The value was obtained using the du –ks command which returned the result "284194 root". This result shows that the sandbox is 284,194 KB and owned by the user root. This converts to approximately 284 MB. The sandbox does not use a majority of the commands and directories in the Solaris operating system which normally takes up two to three gigs. The sandbox only copies the items that are necessary in creating the user environment for SAIT laboratory.

(b) Time it takes to run the server script to create JSS:

This is the amount of time that to create the JSS sandbox on the server. The creation script was run on the server and timed using the ptime command. Table 2 lists four timed runs of the startup script creating the JSS environment on the server. The average time real time to create the sandbox on the server is seven minutes fifty-seven seconds. Although this is a lengthy process, it only needs to be executed initially to create the original default environment on the server or if any changes are made to the server which would reflect inside the JSS filesystem.

	Real (elapsed)	user (CPU time)	sys (CPU time)
Run 1	8:11.818	0.022	0.041
Run 2	7:50.679	0.023	0.045
Run 3	7:37.536	0.023	0.030
Run 4	7:33.577	0.023	0.040

Table 2: JSS Creation Times (startup.sh)

(c) Time it takes to remove JSS:

This is the time that it takes to remove the entire JSS filesystem from the server or the client, which are the same size. The command "rm –Rf" was done on the directory containing the sandbox. Only the startup script executes removal of the JSS environment. This is done when recreating the original sandbox on the server to take in any updates that have been made. Table 3 lists the ptimes recorded for removing the JSS environment. The average amount of time to remove the sandbox is one minute twenty-one seconds. This, in

addition to the creation time of the original default filesystem is the time it takes to make updates the sandbox on the server which is roughly nine minutes 20 seconds.

	Real (elapsed)	User (CPU time)	Sys (CPU time)
Run 1	1:18.959	0.408	4.065
Run 2	1:23.289	0.432	4.327

Table 3: JSS Removal Times

(d) Time it takes to compile client program compare.cpp:
This is the times recorded that it takes to compile the JSS program used on the clients to become jss_user in the sandbox environment. This program only needs to be compiled once on the machine to create the executable on the machine for users. If any updates are made to the actual source code this program will need to be recompiled. Table 4 lists the different information gathered from this timing. The average compilation time was one point nine seconds. The different numbers results are likely due to the other processes simultaneously running on the machine while the program was being executed.

	real (elapsed time)	user (CPU time)	sys (CPU time)
Run 1	2.921	0.006	0.017
Run 2	1.887	0.006	0.015
Run 3	1.887	0.006	0.015
Run 4	1.567	0.006	0.015
Run 5	1.588	0.006	0.025
Run 6	1.819	0.006	0.015

Table 4: Compilation Times for JSS Client Program

(d) Time it takes to create the JSS filesystem on client through rsync:
This time the initial rsync of the client to the server which creates the entire initial environment on the client. This first time takes the longest amount of time due to the fact the entire sandbox is being created and not merely recreating altered portions of the filesystem. This is the only time the entire environment will be copied to the client unless sandbox environment is somehow removed from the system. The average time it takes to copy the JSS environment to the client from the server through rsync is about eight minutes 31 seconds.

	Real (elapsed time)	User (CPU time)	Sys (CPU time)
Run 1	8:30.907	0.010	0.036
Run 2	8:31.339	0.009	0.023

Table 5: Creating the Initial JSS filesystem on the Client through rsync.

(e) Time it takes to log in and out of the program:

When JSS is executed, the time it takes to start the program is in direct relation to the time it takes to update the environment. This is because the more changes that have been made to the sandbox, the longer it will take to update the client environment from the server.

The following tests in Table 6 were conducted to give an average idea of the time it takes to begin the program with no changes to the environment. The large gap in the result times is due to other activity on the network. The average result time when there was little network traffic was four point seven seconds. When there was more network traffic, the average startup time was fifteen seconds. Network traffic is cause by packets going across the network which slow down connections.

Table 7 lists the update times for the JSS environment when a program has been installed into the environment. The average time now jumps up to almost five minutes. This concludes that the more changes that have been made to the environment, the longer it will take to update the sandbox when a user executes the program. However the timings in Table 7 may have been slowed by network traffic due to the time it takes to create the entire environment is only eight minutes 31 seconds.

With no changes to the environment:

	Real (elapsed time)	User (CPU time)	Sys (CPU time)
Run 1	4.132	0.009	0.025
Run 2	4.747	0.009	0.023
Run 3	4.747	0.019	0.022
Run 4	14.757	0.011	0.033
Run 5	5.362	0.010	0.022
Run 6	15.240	0.010	0.022

Table 6: JSS rsync time (no environment changes)

One program (installation) change to the environment:

	Real (elapsed time)	User (CPU time)	Sys (CPU time)
Run 1	4:52.448	0.009	0.034
Run 2	5:02.04	0.010	0.036

Table 7: JSS rsync time (one program installation change)

(f) Testing the lock file of the program:

To ensure that only one user at a time is able to execute the program, the program creates a lock file whenever it is being used. This ensures that only one user has access to the environment at a time. If two or more users simultaneously execute the program, changes they made to the filesystem would reflect upon the sandbox and the environment of the other user. To test that the lock file was working correctly, a program session was executed by the first user.

Following sessions were executed while the first session was in progress. Each of the subsequent sessions displayed an error message to the user stating that the JSS program was already being utilized which meant the lock file was implemented correctly.

(g) Testing the authentication process:

The server authenticates the client username and password through the rsync daemon when the client tries to update its sandbox environment. The daemon is configured through the rsyncd.conf file. Here, the different options are set such as the path directory authenticated users may access and the specific users allowed to connect. To test that the daemon had been correctly configured, different user names and passwords were used to try and authenticate with the rsync server. All username and passwords that had legitimate access were authorized and the rest were denied.

(h) Testing the recreation of the environment:

The user environment on the client machines was tested to ensure that the altered portions were being recreated properly. The first test was conducted by first measuring the size of the newly created filesystem after it has been updated from the server. This is accomplished by using the "du –ks" command on the directory which returns its size in kilobytes. Once changes have been made to the filesystem, such as installing a security tool, the program is closed and then reopened so that the server updates the sandbox again.

The size of the newly updated sandbox was obtained through the before mentioned command and then compared to the previous recorded size. The filesystems were both the same size before and after the user modification which shows that the filesystem is being recreated correctly.

The second way to check the accuracy of the recreated filesystem was to keep a second copy of the untouched JSS environment on the client machine. Using the "diff" command, the entire directory structure of the actual updated sandbox can be compared against the untouched sandbox and would return any differences between the two filesystem. Nothing was returned when using the "diff" command on the filesystem. This result means that there were no differences detected between the two sandboxes.

(i) Testing the Tools in the environment:

To ensure that each of the security tools function in the sandbox, each tool was tested in the JSS filesystem. The tools were tested by being uncompressed, installed and then executed inside the user environment as jss_user.

Many open source freeware tools involved making source code adjustments and installing additional libraries and programs to allow them to work in the sandbox. This installation procedure for programs is standard with all UNIX environments. This is the reason that open source freeware tools are so portable and time consuming to install for the different UNIX operating systems. Each security tool tested functioned correctly inside the sandbox.

7. Conclusion

The first goal of this project was to works towards making the UNIX side of SAIT more a security lab rather than just a computer lab. This is done to promote the three purposes of the laboratory: education, research outreach. This may only be accomplished if the computers remain at a constant environment level allowing users to have a fresh copy of the environment to utilize. This calls for a way to bring machines back to an original default state after users finish their sessions.

The next goal of this project was to find a way to be able to perform security studies on the Ultra V Workstations. This involved creating a way that users would be able to install and execute the collected security tools. This also included researching and configuring a database of security tools for the different decided categories.

The last aim was to protect the overall system from the users. To create this environment some sort of containment method, such as a wrapper or sandbox, needed to be used to only allow users to simulate root access on a certain partition of the machine without affecting the main filesystem. This also required finding a way to update the client environments from the server without having to give users passwords.

To accomplish this project, J's Secure Sandbox was created. This program confines users inside a sandbox environment where they simulate root permissions. This sandbox contains a miniature version of the filesystem that only contains commands and files necessary to install the database of security tools.

The environment is brought back to an original default state by rsyncing from the server. The original untouched copy of the sandbox resides on the server and is replicated to the clients. In this way only altered portions of the JSS filesystem are recreated and all clients are able to maintain identical user environments. As an aid for security studies, different security tools were researched and tested on the Ultra V workstations.

The conclusion to this project is that an environment for users in SAIT laboratory could be created. Through JSS, users are able to install the database of security tools on the machines as the owner of the miniature filesystem, jss_user. Each time JSS is executed, it updates itself from the server which gives users a fresh copy of the environment.

A database of forty security tools were collected and tested. These security tools were configured and executed inside the environment to ensure the tools themselves and the JSS environment worked properly. This was to ensure the sandbox contained the necessary files for a usable environment

The JSS filesystem is a work in progress. New security tools brought into the sandbox may require the installation of new libraries and programs. Also source code adjustments may be necessary to ensure proper execution. Each of these steps was necessary for the collected security tools in this project.

Through JSS and VMWare, the SAIT laboratory has moved towards its goal from evolving from a computer laboratory to a security laboratory.

[A] GLOSSARY

CHOWN The chown utility will set the user ID of the file named by each file to the user ID specified by owner, and, optionally, will set the group ID to that specified by group.

CHMOD Change the permissions mode of a file chmod changes or assigns the mode of a file. The mode of a file specifies its permissions and other attributes. The mode may be absolute or symbolic.

CHROOT Change root directory. The chroot() and fchroot() functions cause a directory to become the root directory, the starting point for path searches for path names beginning with / (slash). The user's working directory is unaffected by the chroot() and fchroot() functions.

DAEMON A program that runs continuously and exists for the purpose of handling periodic service requests that a computer system expects to receive. The daemon program forwards the requests to other programs (or processes) as appropriate.

DIFF Display line-by-line differences between pairs of text files. The diff utility will compare the contents of file1 and file2 and write to standard output a list of changes necessary to convert file1 into file2. This list should be minimal. No output will be produced if the files are identical.

GB A gigabyte (pronounced GIG-a-bite with hard G's) is a measure of computer data storage capacity and is "roughly" a billion bytes. A gigabyte is two to the 30th power, or 1,073,741,824 in decimal notation.

GID A GID (group ID) is a name that associates a system user with other users sharing something in common (perhaps a work project or a department name). It's often used for accounting purposes. A user can be a member of more than one group and thus have more than one GID. Any user using a UNIX system at a given time has both a user ID (UID) and a group ID (GID).

GUI A GUI (usually pronounced GOO-ee) is a graphical (rather than purely textual) user interface to a computer

JSS J’s Secure Sandbox. This program creates a sandbox on the Solaris operating system which contains a miniature filesystem. When the program is launched, the environment allows users to simulate root permissions but confines them to the sandbox.

MAN IN THE MIDDLE ATTACK The "Man In The Middle" or "TCP Hijacking" attack is a well known attack where an attacker sniffs packets from network, modifies them and inserts them back into the network

MEGS Megabyte. As a measure of computer processor storage and real and virtual memory, a megabyte (abbreviated MB) is 2 to the 20th power byte, or 1,048,576 bytes in decimal notation.

MKNOD Mknod makes a directory entry for a special file. These special files are used to interact with character and block devices.

NSCD Name service cache daemon. NSCD is a process that provides a cache for the most common name service requests. It is started up during multi-user boot. The default *configuration-file* /etc/nscd.conf determines the behavior of the cache daemon. See nscd.conf(4). NSCD provides caching for the passwd(4), group(4) and hosts(4) databases through standard libc interfaces, such as gethostbyname(3N), gethostbyaddr(3N), and others. Each cache has a separate time-to-live for its data; modifying the local database (/etc/hosts, and so forth) causes that cache to become invalidated within ten seconds. Note that the shadow file is specifically not cached. getspnam(3C) calls remain uncached as a result.

PROC Virtual filesystem which provides the capability to inspect processes.

PTIME Time the command, like time(1), but using microstate accounting for reproducible precision. Unlike time(1), children of the command are not timed.

RSYNC Rsync is an open source utility that provides fast incremental file transfer. Rsync is freely available under the GNU General Public License.

SYMLINK A new directory entry (link) for the file specified by source_file, at the destination path specified by target. If target is not specified, the link is made in the current directory. This first synopsis form is assumed when the final operand does not name an existing directory; if more than two operands are specified and the final is not an existing directory, an error will result.

TIME Time a simple command.

TRUSS Trace system calls and signals. The truss utility executes the specified command and produces a trace of the system calls it performs, the signals it receives, and the machine faults it incurs. Each line of the trace output reports the fault, signal name or the system call name with its arguments and return value(s). System call arguments are displayed symbolically when possible using defines from relevant system headers; for any path name pointer argument, the pointed-to string is displayed. Error returns are reported using the error code names described in intro(3).

UID A unique positive integer that identifies the user to the operating system. Two users that have the same UID are considered the same user to the operating system.

ZLIB DOUBLE FREE BUG There is a security vulnerability in zlib 1.1.3 that can be exploited by providing a specially crafted invalid compressed data stream to zlib's decompression routines that results in zlib attempting to free the same memory twice. On many systems, freeing the same memory twice will crash the application. Such "double free" vulnerabilities can be used in denial-of-service attacks, and it is remotely possible that the vulnerability could be exploited in some application to execute arbitrary code with that application's permissions. There have been no reports of any exploitation of this problem, but nevertheless the vulnerability exists.

[B] REFERENCES

[1] Adler, Mark. "Zlib Advisory 2002-03-11."
http://www.gzip.org/zlib/advisory-2002-03-11.txt March 14, 2002.

[2] "Advanced Research Corporation."
www-arc.com March 31, 2002

[3] "Arpwatch." Security Focus Online.
http://online.securityfocus.com/tools/142 March 31, 2002

[4] "Authentication Tools." U.S. Department of Energy.
http://ciac.llnl.gov/ciac/ToolsUnixAuth.html#Crack March 31, 2002

[5] Baker, Ted, Ivo Desmedt. "SAIT Security and Assurance in Information Technology Laboratory." http://www.sait.fsu.edu December 2001

[6] "Benefits of Using Deep Freeze."
http://www.deepfreezeusa.com/benefits.htm#2 March 31, 2002

[7] "Big Brother is Watching." Big Brother.
http://bb4.com/ March 31, 2002

[8] Bogdanov, Marion. "Creating the SAIT Laboratory."
Department of Computer Science, Florida State University. December 2001.

[9] Casillas, Juan. "Jail Chroot Project."
http://www.gsyc.inf.uc3m.es/~assman/jail/ October 29, 2001

[10] "Cert Advisory CA-1993-14 Internet Security Scanner."
http://www.cert.org/advisories/CA-1993-14.html September 19, 1997

[11] "chmod." User Commands, chmod(1).
February 1, 1995.

[12] "chown." User Commands Man Page, chown(1).
June 1, 1998.

[13] "chroot." Maintenance Commands, chroot(1M).
March 20, 1998

[14] "CISE Computer & Information Science & Engineering." University of Florida.
http://www.cise.ufl.edu/depot/new/irix6.5-tcp_wrappers-7.6.shtml March 31, 2002

[15] "Cryptographic Checksums." U.S. Department of Energy
http://ciac.llnl.gov/ciac/ToolsUnixSig.html March 31, 2002

[16] "Daemon." SearchSolaris.com
http://searchsolaris.techtarget.com/sDefinition/0,,sid12_gci211888,00.html
December 11, 1999.

[17] "des.tar.gz." Cerias FTP site.
ftp.cerias.purdue.edu/pub/tools/unix/crypto/des-aain/des.tar.gz March 31, 2002

[18] "diff." User Commands, diff(1);
December 20, 1996.

[19] "Download the Nessus Security System Scanner for POSIX systems." Nessus.
http://www.nessus.org/posix.html March 31, 2002

[20] Elson, Jeremy. "TCPFlow – A TCP Flow Recorder."
http://www.circlemud.org/~jelson/software/tcpflow June 8, 20001

[21] "Gigabyte." SearchStorage.com
http://searchstorage.techtarget.com/sDefinition/0,,sid5_gci212194,00.html
July 31, 2001

[22] "GID." SearchWin2000.com
http://searchwin2000.techtarget.com/sDefinition/0,,sid1_gci212189,00.html
October 18, 1999.

[23] "The GNU Privacy Guard."
http://www.gnupg.org/ January 22, 2002

[24] "GUI." SearchWebServices.com
http://searchwebservices.techtarget.com/sDefinition/0,,sid26_gci213989,00.html
July 30, 2002.

[25] "Home Tripwire.org." Tripwire, Inc.
http://www.tripwire.org March 31, 2002

[26] "Index of /CERT/Software."
http://www.ja.net/CERT/Software/ March 31, 2002

[27] "Intelligent Security." Psionic Technologies.
www.psionic.com March 31, 2002

[28] "John the Ripper – Password Cracker."
http://user-mode-linux.sourceforge.net March 31, 2002

[29] "ls." User Command Man Page, ls(1).
May 5, 1997.

[30] "MDCrack Brute Force Your Hashes."
http://membres.lycos.fr/mdcrack/
March 31, 2002.

[31] "Megabyte." Whatis.com
http://whatis.techtarget.com/definition/0,,sid9_gci212542,00.html
March 31, 2002.

[32] "mknod." Maintenance Commands, mknod(1M).
September 16, 1996.

[33] "Muffin World Wide Web Filtering System."
http://muffin.doit.org/ March 31, 2002

[34] Nemeth, Evi, et al. "UNIX System Administration Handbook."
Prentice Hall: New Jersey 2001

[35] "Network Monitoring Tools." U.S. Department of Energy
http://ciac.llnl.gov/ciac/ToolsUnixNetMon.html March 31, 2002

[36] "NMAP Network Security Scanner." NMAP.
www.insecure.org/nmap/nmap_download.html March 19, 2002

[37] "NSCD." AnswerBook2
http://sundocs.princeton.edu:8888/ab2/coll.40.5/REFMAN1M/@Ab2PageView/142571
July 17, 1997.

[38] Provos, Niels. "ScanSSH."
http://www.monkey.org/~provos/scanssh/ March 31, 2002

[39] "ptime." User Commands, proc(1).
November 17, 1999.

[40] "Queso 6.6."
http://users.iol.it/alfier/soft/queso.html March 31, 2002

[41] "Raw IP Networking FAQ."
http://www.whitefang.com/rin/rawfaq.html#8 April 20, 2002

[42] "Rsync."
http://samba.anu.edu.au/rsync March 31, 2002

[43] "SAINT Vulnerability Assessment Tools."
www.wwdsi.com/saint March 28, 2002

[44] "Sandbox." SearchDatabase.com

http://searchsecurity.techtarget.com/sDefinition/0,,sid14_gci283994,00.html
July 3, 2001.

[45] "Security Tools: File Integrity Checker."
http://www.mycert.mimos.my/resource/fic.htm March 31, 2002

[46] "Sniffers."
http://www.solaris4you.dk/sniffersSS.html March 31, 2002

[47] "Snort The Open Source Network Intrusion Detection System."
www.snort.org March 23, 2002

[48] "Socks."
http://www.socks.nec.com/cgi-bin/download.pl March 31, 2002.

[49] "System Monitoring Tools." U.S Department of Energy.
http://ciac.llnl.gov/ciac/ToolsUnixSysMon.html March 31, 2002

[50] "TCPDump/Libpcap."
www.tcpdump.org February 4, 2002

[51] "time." User Commands, time(1).
February 1, 1995

[52] "TITAN 4.0 BETA2."
www.fish.com/titan/TITAN_documentation.html August 31, 2002

[53] "The User-Mode Linux Kernel Home Page."
http://user-mode-linux.sourceforge.net March 31, 2002

[54] "Virtual Computing Throughout the Enterprise." VMWare.
March 31, 2002.

[55] "Whisker Information, Scripts and Updates."
www.wiretrip.net/rfp/p/doc.asp/i1/d21.htm March 29, 2002

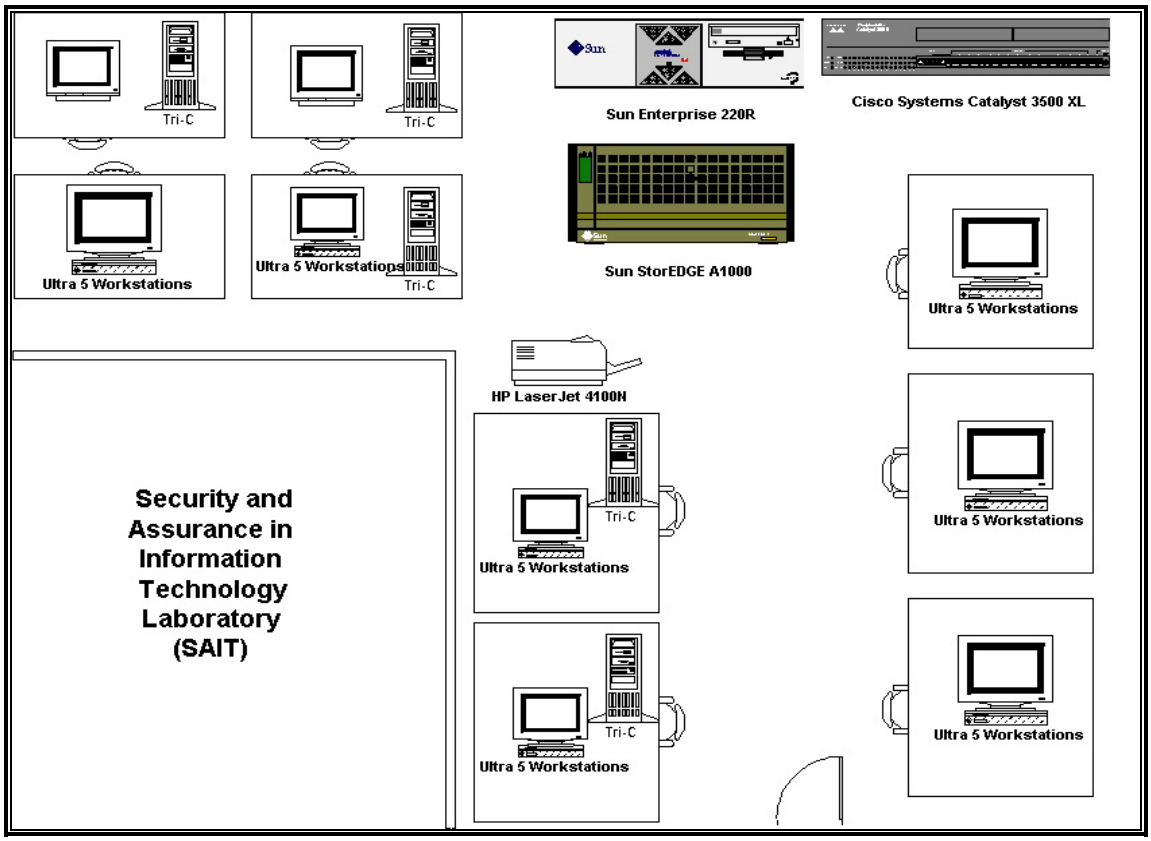
[56] "Wrapper." SearchDatabase.com
http://searchdatabase.techtarget.com/sDefinition/0,,sid13_gci213388,00.html
August 4, 2001.

[57] Yasinsac, Alec. "Active Protection of Trusted Security Services."
Department of Computer Science, Florida State University, Jan 2000.

[58] Yasinsac, Alec, Jennifer Frazier, Marion Bogdanov. "Developing an Academic Security Laboratory." Department of Computer Science, Florida State University. January 2002.

[C] SAIT Layout

This is a layout of the SAIT laboratory [42].



[D] Tools with their Description and Installation Directions

Firewall - SocksV5

Socksv5 allows UNIX hosts behind a firewall to gain full access to the Internet without requiring direct IP reachability. socksv5 requires a SOCKS daemon running on a host that can communicate directly with hosts behind the firewall and with hosts on the Internet.

Program Name:	SocksV5
Program Website:	http://www.socks.nec.com/cgi-bin/download.pl
Program Source Code:	socks5-v1.0r11.tar.gz
Program Requirements:	No Special Requirements
Installation Steps:	(1) ./configure
	(2) make install
	(3) make
To Run:	cd to socks5-vs.0r11/server and type: ./socks5

Intrusion Detection – Sniffer - TCPDUMP

Program used to capture packets.

Program Name:	TCPDUMP
Program Website:	www.tcpdump.org
Program Source Code:	tcpdump-3.6.2.tar.gz
Program Requirements:	libpcap-0.6.2.tar.gz
	flex-2.5.4.tar.gz
	bison-1.29.tar.gz
Installation Steps for tcpdump:	(1) ./configure
	(2) make
	(3) make check
	(4) make install
To Run:	./tcpdump
Front End Tools:	TCPSlice

Firewall - Muffin

A filtering proxy server for the World Wide Web.

Program Name:	Muffin
Program Website:	http://muffin.doit.org/
Program Source Code:	muffin-0.9.3a.tar.gz
	muffin.jar
Program Requirements:	JAVA
Installation Steps:	(1) ./configure
	(2) cp muffin.jar to /usr/local/src and muffin-0.9.3a/src
	(3) make install
	(4) ln -s java1.2/jre /usr/jre
	(5) ln -s /usr/java1.2/jre/bin/sparc/native_threads /usr/bin/sparc/native_threads
To Run:	In muffin-0.9.3a/src: ./muffin
Web Configuration:	(1) In Netscape, go to edit, preferences, advanced, proxies
	(2) Next to Manual Proxy, click view
	(3) In HTTP Proxy, enter the hostname and IP address that the proxy is running on, Muffin outputs this info when started

Intrusion Detection – Sniffer - TCPFlow

TCPFlow is a program that captures data transmitted as part of TCP connections (flows), and stores it in a way that is convenient for protocol analysis or debugging.

Program Name:	TCPFlow
Program Website:	http://www.circlemud.org/~jelson/software/tcpflow/
Program Source Code:	tcpflow-0.12.tar.gz
Program Requirements:	Libpcap
Installation Steps:	(1) ./configure --with-pcap=[libpcap path]
	(2) make
	(3) make install
To Run:	./tcpflow

Intrusion Detection – Sniffer - Sniffit

Sniffit is a robust non-commercial network protocol analyzer or packet sniffer. A packet sniffer basically listens to network traffic and produces analysis based on the traffic and/or translates packets into some level of human readable form.

Program Name:	Sniffit
Program Website:	http://www.solaris4you.dk/sniffersSS.html
Program Source Code:	sniffit.0.3.5.tar.gz
Program Requirements:	Libpcap
Installation Steps:	(1) Install Libpcap (current v 0.6.2)
	(2) In the files: configure, configure.in, makefile, makefile.in, using a VI editor, you need to use this command: :1,$s/libpcap-0.3/libpcap-0.6.2/g
	(3) ./configure
	(4) In sniffit.0.3.5.c (lines 1442,1445) change 0 to jss_user GID and UID
	(5) make
To Run:	./sniffit [-xdabvnN] [-P proto] [-A char] [-p port] [-l sniflen] [-L loglevel] [-F snifdevice] [-M plugin] (-t | -s) | -c]

Host Based Intrusion Detection - Watcher

The Watcher package by Kenneth Ingham. A configurable and extensible system monitoring tool that issues a number of user-specified commands, parses the output, checks for items of significance, and reports them to the system administrator.

Program Name:	Watcher
Program Website:	http://ciac.llnl.gov/ciac/ToolsUnixSysMon.html#Watcher
Program Source Code:	Watcher.tar.Z
Program Requirements:	No Special Requirements
Installation Steps:	(1) In Makefile, change line 5 to be: SYS=SYSV In Makefile on line 6 add: CC=gcc
	(2) Create a "watcherfile"
	(3) To use this example file, you will also need a "daemons" file
To Run:	./watcher

Host Based Intrusion Detection - TCP Wrappers

PortSentry is part of the Abacus Project suite of security tools. It is a program designed to detect and respond to port scans against a target host in real-time. It runs on TCP and UDP sockets and works on most UNIX systems. Advanced stealth detection modes are available under Linux only and detect SYN, FIN, NULL, XMAS, and Oddball packet scans. All modes support real-time blocking and reporting of violations.

Program Name:	TCP Wrappers
Program Website:	http://www.cise.ufl.edu/depot/new/irix6.5-tcp_wrappers-7.6.shtml
Program Source Code:	tcp_wrappers_7.6.tar.gz
Program Requirements:	No Special Requirements
Installation Steps:	(1) Uncomment line 44 in the Makefile
	(2) Type "make sunos5 CC=gcc"
To Run:	Need to modify the inetd.conf file with a command such as: tftp dgram udp wait root /usr/etc/tcpd in.ftpd -s /tftpboot

Host Based Intrusion Detection - Tripwire

Tripwire is a tool that checks to see what has changed on your system. The program monitors key attributes of files that should not change, including binary signature, size, expected change of size, etc.

Program Name:	Tripwire
Program Website:	http://www.tripwire.org
Program Source Code:	Tripwire-1.3.1-1.tar.gz
Program Requirements:	No Special Requirements
Installation Steps:	(1) make
To Test:	Make test
To Run:	(1) mkdir /usr/local/bin/tw
	(2) Edit the tw_ASR_1.3.1_src/config/tw.config
	(3) copy tw_ASR_1.3.1_src/config/tw.config to /usr/local/bin/tw
	(4) cd to tw_ASR_1.3.1_src/src and type: ./tripwire -initialize
	(5) mkdir /var/tripwire
	(6) cp .databases/tw.db_[hostname] to /var/tripwire
	(7) cd tw_ASR_1.3.1_src/src and type: ./tripwire [-option] [file(s)]

Host Based Intrusion Detection - L5

L5 simply walks down Unix or DOS filesystems, sort of like "ls -R" or "find" would, generating listings of anything it finds there. It tells you everything it can about a file's status, and adds on the MD5 hash of it. Its output is rather "numeric", but it is a very simple format and is designed to be post-treated by scripts that call L5.

Program Name:	L5
Program Website:	http://www.mycert.mimos.my/resource/fic.htm
Program Source Code:	L5.tar.gz
Program Requirements:	No Special Requirements
Installation Steps:	(1) Change line 12 in Makefile to be gcc instead of cc
	(2) make solaris
To Test:	(1) echo foofoo | ./l5 should produce: -STANDARD INPUT-//X - - -/- 7 - 1vnGam6fDhYM5zgofZB2Ei
To Run:	l5 [filename] | l5

Network Intrusion Detection - GABRIEL

Gabriel is a SATAN detector, similar to Courtney. While it is only available for Sun platforms, it is written entirely in C, and comes pre-built.

Program Name:	GABRIEL
Program Website:	http://ciac.llnl.gov/ciac/ToolsUnixNetMon.html#Gabriel
Program Source Code:	gabriel-1.0.tar.gz
Program Requirements:	No Special Requirements
Installation Steps:	Program Comes Precompiled
To Test:	./gabriel_tester
To Run:	(On the Server) ./gabriel_server
	(On the Client) ./gabriel_client

Network Intrusion Detection - Argus

Argus is a generic IP network transaction auditing tool that has allowed Carnegie Mellon University's Software Engineering Institute to perform a number of powerful network management tasks that are currently not possible using commercial network management tools. It requires the libpcap and tcp_wrappers packages, both of which are available in the same directory as the Argus software.

Program Name:	Argus
Program Website:	http://ciac.llnl.gov/ciac/ToolsUnixNetMon.html#Argus
Program Source Code:	argus-1.8.1.tar.gz
Program Requirements:	Libpcap
	TCP Wrappers
Installation Steps:	(1) Ensure you install TCP Wrapper and Libpcap inside the argus-1.8.1 directory
	(2) ./configure
	(3) make
To Run:	(1) cd argus-1.8.1/bin
	(2) ./argus_"libpcapinterfacename" which should be argus_dlpi

Network Intrusion Detection - Courtney

Courtney is a program devised by Lawrence Livermore Laboratory scientist Marvin Christensen that will alert people that someone has probed their computers using Satan. Christensen named Courtney after his 18-month-old daughter.

Program Name:	Courtney
Program Website:	http://ciac.llnl.gov/ciac/ToolsUnixNetMon.html#Courtney
Program Source Code:	courtney-1.3.tar.gz
Program Requirements:	TCPDump
	Libpcap (Also needed by TCPDump)
Installation Steps:	(1) Place the directories Libpcap and TCPDump in the same directory with the Courtney executable (courtney.pl)
	(2) Edit Courtney's path (line 55 in courtney.pl) to include the directories. /tcpdump-3.6.2 and. /libpcap-0.6.2
To Run:	./courtney &

Network Intrusion Detection - Snort

There are three main modes in which Snort can be configured: sniffer, packet logger, and network intrusion detection system. Sniffer mode simply reads the packets off of the network and displays them for you in a continuous stream on the console. Packet logger mode logs the packets to the disk. Network intrusion detection mode is the most complex and configurable configuration, allowing Snort to analyze network traffic for matches against a user defined rule set and perform several actions based upon what it sees.

Program Name:	Snort
Program Website:	www.snort.org
Program Source Code:	snort-1.8.1.tar.gz
Program Requirements:	Libpcap
Installation Steps:	(1) ./configure
	(2) make
	(3) make install
	(3) mkdir /var/log/snort
To Run:	./snort -[options] [filter]

Media Prevention - Scrub

The SCRUB utility is a program for sanitizing UNIX files or disk drives. Writes patterns on disk/file. Writes patterns on special files (i.e. raw disk devices) or regular files to reduce the possibility of someone retrieving the data. There are up to six passes, each of which fills the disk with a particular pattern: 0. 0x00 1. 0xff 2. 0xaa 3. random (if option specified) 4. 0x55 5. verify pattern 4

Program Name:	Scrub
Program Website:	http://doe-is.llnl.gov/SecRes/DOECustomTools.html
Program Source Code:	scrub-1.3.tar.gz
Program Requirements:	No Special Requirements
Installation Steps:	(1) make
To Run:	./scrub -d [filename]

Network Intrusion Detection - Swatch

The Swatch package by Stephen Hansen and Todd Atkins. A system for monitoring events on a large number of systems. Modifies certain programs to enhance their logging capabilities, and software to then monitor the system logs for ``important'' messages.

Program Name:	Swatch
Program Website:	http://ciac.llnl.gov/ciac/ToolsUnixSysMon.html#Swatch
Program Source Code:	swatch-3.0.b1.tar.gz
Program Requirements:	Bit-Vector-6.1
	Date-Calc-5.0
	File-Tail-0.98
	Time-HiRes-01.20
Installation Steps:	(1) Install the Modules above in the following order: Bit-Vector, Date-Calc, Time-HiRes, File-Tail
	(2) perl Makefile.pl
	(3) make
	(4) make test
	(5) make install
	(6) make realclean
To Run:	./swatch

Authentication & Encryption – Encryption & Decryption - DES

Program Name:	DES
Program FTP site:	ftp.cerias.purdue.edu/pub/tools/unix/crypto/des-aain/des.tar.gz
Program Source Code:	des.tar.gz
Program Requirements:	No Special Requirements
Installation Steps:	(1) open /des/lib/alo-getpass.c and to the top add: #include [fcntl.h]; #include [sys/stat.h]
	(2) make CC=gcc
To Run:	(1) cd to des/prog
	(2) To encrypt: ./des -e testdata > [file]
	(3) To decrypt: ./des -d file and redirect to stout/file

Network Intrusion Detection - Arpwatch

Arpwatch is a tool that monitors ethernet activity and keeps a database of ethernet/ip address pairings.

Program Name:	ArpWatch
Program Website:	http://online.securityfocus.com/tools/142
Program Source Code:	arpwatch-2.1a4.tar.gz
Program Requirements:	Libpcap
Installation Steps:	(1) Edit BINDEST and MANDEST in Makefile.in to user selected path
	(2) Ensure that libpcap has the same parent directory
	(3) ./configure
	(4) In Makefile (lines 113,114,116,118) and Makefile.in (lines 112,113,116,118) change bin to jss_user
	(5) make
	(6) make install
To Run:	./arpwatch -d

Authentication & Encryption – Hashes - Snefru

The source code and documentation for the Snefru message digest function (Xerox's Secure Hash Function).

Program Name:	Snefru
Program Website:	http://ciac.llnl.gov/ciac/ToolsUnixSig.html#Snefru
Program Source Code:	snefru-2.5a.tar.gz
Program Requirements:	No Special Requirements
Installation Steps:	(1) gcc -o snefru snefru.c
To Test:	./testSnefru
To Run:	echo [what you want to hash] | ./snefru >> [output file]

Auditing Intrusion Detection - Portsentry

PortSentry is part of the Abacus Project suite of security tools. It is a program designed to detect and respond to port scans against a target host in real-time. It runs on TCP an d UDP sockets and works on most UNIX systems. Advanced stealth detection modes are availab le under Linux only and detect SYN, FIN, NULL, XMAS, and Oddball packet scans. All modes s upport real-time blocking and reporting of violations

Program Name:	Portsentry
Program Website:	www.psionic.com
Program Source Code:	portsentry-1.1.tar.gz
Program Requirements:	No Special Requirements
Installation Steps:	1. In the makefile:(A) put a # in front of CC=gcc (B) Take out the # in line 26
	2. On line 59 in portsentry.c, on line 59 change 0 to be 101
	3. make solaris
	4. make install
To Run:	./portsentry (option)

Authentication & Encryption - GPG

GnuPG is a complete and free replacement for PGP. Because it does not use the patented IDEA algorithm, it can be used without any restrictions. GnuPG is a RFC2440 (OpenPGP) compliant application.

Program Name:	GPG
Program Website:	http://www.gnupg.org/
Program Source Code:	gnupg-1.0.6.tar.gz
Program Requirements:	No Special Requirements
Installation Steps:	(1) ./configure
	(2) make
	(3) make install
To Run:	(1) ./gpg --gen-key
	(2) Follow Intructions

Big Brother monitors System and Network-delivered services for availability Big Brother is designed to let anyone see how their network is doing in near real-time, from any web browser, anywhere.

Program Name:	Big Brother
Program Website:	http://bb4.com/
Program Source Code:	bb-1.8c1.tar.gz
Program Requirements:	Apache
Installation Steps:	(1) mkdir /home/jss_user; cd /home/jss_user
	(2) mv bb18c1 /home/jss_user Make directoires if necessary
	(3) cd install
	(4) ./bbconfig sunos
	(5) make directories: /home/www/httpd/cgi-bin
	(6) Yes to License; No to run as root; jss_user for user ID of BB; No to old-style directory structure; Yes to FQDN; [Enter] through the rest of the options
	(7) cd ../src;
	(8) make
	(9) make install
	(10) vi bb18c1/user/etc/bb-hosts add the line: [ip address] [machine name] #BBDISPLAY BBPAGER BBNET (ex. 192.168.1.1 machine.cs.fsu.edu) Comment out all other lines in file
	(11) make install
To Run:	./runbb.sh

Miscellaneous – Testing Tool - TCPReplay

TCPReplay is a tool to replay saved tcpdump files at arbitrary speeds. This program was written in the hopes that a more precise testing methodology might be applied to the area of network intrusion detection, which is still a black art at best. Many NIDSs fare poorly when looking for attacks on heavily-loaded networks. Tcpreplay allows you to recreate real network traffic from a real network for use in testing.

Program Name:	TCPReplay
Program Website:	http://www.dbaseiv.net/info/tools/tcpreplay/
Program Source Code:	tcpreplay-1.0.tar.gz
Program Requirements:	TCPDUMP
	Most recent version of Libpcap (an older version comes with the program)
Installation Steps:	(1) At the end of line 14 of the Makefile.in add: -lsocket -lnsl -lresolv
	(2) Ensure line 19 of the makefile is using the correct version of libpcap.
	(3) ./configure
	(4) make
	(5) make install
To Run:	Collect packets with tcpdump using the command: ./tcpdump -w [filename]
	Copy this file to the tcpreplay directory: ./tcpreplay < [filename]

Auditing Intrusion Detection - Antiroute

Prevents and logs UDP-based route tracking Antiroute listens on ports used in UDP-based route tracking and determines the IP address, source port and distance (in hops) of the host from which the trace is being performed.

Program Name:	Antiroute
Program Website:	http://www.lovric.net/antiroute
Program Source Code:	antiroute-1.1.tar.gz
Program Requirements:	No Special Requirements
Installation Steps:	(1) gcc -O3 -lsocket -o antiroute antiroute.c
To Run:	./antiroute -o

Vulnerability Management – Port Scanner - NMAP

Nmap ("Network Mapper") is an open source utility for network exploration or security auditing. It was designed to rapidly scan large networks, although it works fine against single hosts. Nmap uses raw IP packets in novel ways to determine what hosts are available on the network, what services (ports) they are offering, what operating system (and OS version) they are running, what type of packet filters/firewalls are in use, and dozens of other characteristics.

Program Name:	NMAP
Program Website:	www.insecure.org/nmap/nmap_download.html
Program Source Code:	nmap-2.53.tar.gz
Program Requirements:	flex-2.5.4.tar.gz
	bison-1.29.tar.gz
Installation Steps:	(1) ./configure
	(2) make
	(3) make install
To Run:	./nmap

Authentication & Encryption – Hashes - MD5

MD5 reads data and calculates a cryptographic "checksum" that is, as far as anyone knows today, very hard to duplicate. Just as traditional checksums are used to gain confidence that a file has not been accidentally modified, say when transmitted over a phone line, MD5 is used to gain confidence that a file has not been *intentionally* modified, say to install a Trojan horse.

Program Name:	MD5
Program Website:	http://ciac.llnl.gov/ciac/ToolsUnixSig.html#Md5
Program Source Code:	MD5.tar.Z
Program Requirements:	No Special Requirements
Installation Steps:	(1) make
To Test:	(time trial) ./md5 -t
	(test script) ./md5 -x
To Run:	./md5 [-s(string)| filename(s)]

PGP

PGP® or Pretty Good Privacy® is a powerful cryptographic product family that enables people to securely exchange messages, and to secure files, disk volumes and network connections with both privacy and strong authentication.

Program Name:	PGP Command Line Freeware
Program Website:	http://web.mit.edu/network/pgp.html
Program Source Code:	PGPcmdln_6.5.8.Sol_FW.tar.gz
Program Requirements:	No Special Requirements
Installation Steps:	Program comes precompiled
To Run:	./pgp [option]

Vulnerability Management – Port Scanner - Whisker

Whisker scan scripts are to be 'fed' into whisker to perform various vulnerability checking. You can browse the scripts and snippets below, combining them to your own personal design.

Program Name:	Whisker
Program Website:	www.wiretrip.net/rfp/p/doc.asp/i1/d21.htm
Program Source Code:	whisker-1.4.tar.gz
Program Requirements:	1. Perl
Installation Steps:	Program Comes Precompiled
To Run:	./whisker (option)

Vulnerability Management – Password Assessment - John the Ripper

John the Ripper is a fast password cracker, currently available for many flavors of Unix (11 are officially supported, not counting different architectures), DOS, Win32, and BeOS. Its primary purpose is to detect weak Unix passwords, but a number of other hash types are supported as well.

Program Name:	John the Ripper
Program Website:	http://www.openwall.com/john
Program Source Code:	john-1.6.tar.gz
Program Requirements:	No Special Requirements
Installation Steps:	(1) cd into john-1.6/src
	(2) make solaris-sparc-gcc
To Run:	(1) cd ../run (out of john-1.6/src to john-1.6/run)
	(2) ./john [option] [password-files]

Vulnerability Management – Password Assessment - Crack

The Crack program by Alex Muffett. A password-cracking program with a configuration language, allowing the user to program the types of guesses attempted.

Program Name:	Crack
Program Website:	http://ciac.llnl.gov/ciac/ToolsUnixAuth.html#Crack
Program Source Code:	crack5.0.tar.gz
Program Requirements:	No Special Requirements
Installation Steps:	(1) In c50a/src/, change the Makefile for lib/Makefile (line 14) and util/Makefile (line 13) and libdes/Makefile (line 38) to CC=gcc instead of CC=$(XCC)
	(2) Make sure line 16 in Crack contains correct path
	(3) Make sure lines 50,51 are not commented and that all others for the compiler are commented in Crack file
	(4) ./Crack -makeonly
	(5) ./Crack -makedict
To Run:	./Crack [options] [bindir] [[-fmt format] files]

Vulnerability Management – Password Assessment - MDCrack

A brute force tool which "attempts" to crack MD4 and MD5 hashes.

Program Name:	MDCrack
Program Website:	http://membres.lycos.fr/mdcrack/
Program Source Code:	mdcrack-1.2.tar.gz
Program Requirements:	No Special Requirements
Installation Steps:	(1) Comment out lines 55-56 and 58-59 in Makefile
	(2) make big install
To Run:	mdcrack -R [file containing hash] such as mdcrack-1.2/sample/sam_sample.txt which is included

Vulnerability Management – Port Scanner - Strobe Scanning Tool

Strobe displays all active listening TCP port on remote hosts. It uses an algorithm which efficiently uses network bandwidth.

Program Name:	Strobe
Program Website:	Due to STROBE getting financial backing it is no longer available online.
Program Source Code:	strobe-1.03.tar.gz
Program Requirements:	No Special Requirements
Installation Steps:	1. cp strobe.1 /usr/local/man/man1
	2. cp strobe.services /usr/local/lib
	3. cp strobe /usr/local/bin
	4. Ensure that line 10 of the Makefile is commented out and that line 11 is not commented out
	5. make install
To Test:	./strobe localhost
To Run:	./strobe

Vulnerability Management – Port Scanner - Internet Security Scanner

Internet Security Scanner (ISS) is the first multi-level security scanners available to the public. It was designed to be flexible and easily portable to many unix platforms and do its job in a reasonable amount of time.

Program Name:	ISS
Program Website:	http://www.cert.org/advisories/CA-1993-14.html
Program Source Code:	iss13.tar.gz
Program Requirements:	Latest Version of Strobe
Installation Steps:	(1) End of line 6 in Makefile add: -lsocket -lnsl -lresolv
	(2) make CC=gcc
To Run:	./iss [option] #1Host #2Host Unless otherwise specified, all information is sent to ISS.log

Vulnerability Management – Port Scanner - ScanSSH

Scanssh scans the given addresses and networks for running SSH servers. It will query their version number and displays the results in a list.

Program Name:	ScanSSH
Program Website:	http://www.monkey.org/~provos/scanssh/
Program Source Code:	scanssh-1.6b.tar.gz
Program Requirements:	Libpcap
Installation Steps:	(1) ./configure
	(2) make
To Run:	./scanssh [ipaddress]

Vulnerability Mangement – Security Assessment Scanner - COPS

The Computer Oracle and Password System (COPS) package from Purdue University. Examines a system for a number of known weaknesses and alerts the system administrator to them; in some cases it can automatically correct these problems.

Program Name:	COPS
Program Website:	http://ciac.llnl.gov/ciac/ToolsUnixSysMon.html#COPS
Program Source Code:	cops.1.04.tar.gz
Program Requirements:	No Special Requirements
Installation Steps:	(1) In makefile, change line 41 to: CC=/usr/local/bin/gcc
	(2) add to the end of lines 69,72,75,78,81,84,87,100,103,106: -ldl
	(3) reconfig
	(4) make
	(5) make install
To Run:	./cops -v -s .
	To see the results, check the "result.process ID number" file

Vulnerability Management – Lockdown - Titan

Titan is a collection of programs, each of which either fixes or tightens one or more potential security problems with a particular aspect in the setup or configuration of a Unix system.

Program Name:	Titan
Program Website:	www.fish.com/titan/TITAN_documentation.html
Program Source Code:	titan-4.0BETA1.tar.gz
Program Requirements:	No Special Requirements
Installation Steps:	1. In the directory titan/lib, change is_root (line 24) to be jss_user instead of root
	2. ./Titan-Config -I
	3. When prompted type "y"
To Run:	./Titan (option)

Miscellaneous – Front End - TCPSlice

Front End tools for TCPDump.

Program Name:	TCPSlice
Program Website:	http://jarok.cs.ohiou.edu/software/tcptrace/useful.html
Program FTP Site:	ftp://ftp.ee.lbl.gov/tcpslice-1.2a1.tar.gz
Program Source Code:	tcpslice-1.2a1.tar.gz
Program Requirements:	TCPDUMP
	libpcap
Installation Steps:	(1) Edit Makefile to ensure that BINDEST & MANDEST both have the correct path for TCPDUMP (most likely /usr/local/tcpdump)
	(2) ./configure
	(3) make
	(4) make install
	(5) make install-man
To Run:	Start running TCPDUMP - in the TCPDUMP directory ./tcpdump
	Start running TCPSlice - ./tcpslice

Miscellaneous – Front End - Merlin

Program Name:	Merlin
Program Website:	http://ciac.llnl.gov.ciac/ToolsUnixSysMon.html#Merlin
Program Source Code:	merlin-1.0.tar.gz
Program Requirements (At least one of the following):	COPS
	TIGER
	Crack
	Tripwire
Installation Steps:	(1) ./merlin-config
	(2) If merlin location is correct, hit the "enter" key, else type "pwd" to get Merlin's path
	(3) If netscape location is correct, hit the "enter" key, else type "which netscape" to get Netscape's path
	(4) make
To Run:	./merlin
To Configure:	(1) Click on "Configuration"
	(2) Enter the path to the selected tool
	(3) Click the "Make Changes" button

Vulnerability Management – Lockdown - Chkacct

CHKACCT checks the files in your Unix account for security problems. chkacct(1) will present each problem to you along with a short explanation as to why it is a danger. You will then be asked if you wish to ignore the problem, see more information about the problem, or have chkacct(1) fix the problem for you.

Program Name:	Chkacct
Program Website:	http://www.ja.net/CERT/Software/chkacct
Program Source Code:	chkacct-1.2.tar.gz
Program Requirements:	No Special Requirements
Installation Steps:	(1) make
To Run:	./chkacct

Vulnerability Management – Lockdown - CheckXusers

This script checks for people logged on to a local machine from insecure X servers. It is intended for system administrators to check up on whether users are exposing the system to unacceptable risks. Like many commands, such as finger(1), checkXusers could potentially be used for less honorable purposes. checkXusers should be run from an ordinary user account, not root. It uses kill which is pretty dangerous for a superuser. It assumes that the netstat command is somewhere in the PATH.

Program Name:	CheckXusers
Program Website:	http://www.ja.net/CERT/Software/checkxusers
Program Source Code:	checkxusers.tar.gz
Program Requirements:	No Special Requirements
Installation Steps:	(1) Make sure the path at the top of the script is correct for perl (this machine /usr/bin/perl)
	(2) Make sure the program is executable (chmod +x checkXusers)
To Run:	./checkXusers

Vulnerability Management – Security Assessment Scanner - TIGER

The tiger package of system monitoring scripts. Similar to COPS in what they do, but significantly more up to date, and easier to configure and use.

Program Name:	TIGER
Program Website:	http://ciac.llnl.gov/ciac/ToolsUnixSysMon.html#Tiger
Program FTP site:	ftp://ftp.cerias.purdue.edu/pub/tools/unix/scanners/tiger
Program Source Code:	tiger-2.2.4p1.tar.gz
Program Requirements:	No Special Requirements
Installation Steps:	Program Comes Precompiled.
To Run:	./tiger

[E] User and Maintenance Document

SERVER – For JSS to be updated to the latest version of commands and libraries on the server, startup.sh needs to be executed.

To add additional commands to JSS, the command needs to be placed into commands_list and then find_lib.sh needs to be executed.

To add additional directories or specific files, it needs to be placed into dir_file_list and then find_dir.sh needs to be executed.

To add additional dot files or compressed files, it needs to be placed into dot_files and then find_dot.sh needs to be executed.

Additional symlinks to be placed into JSS need to be added into mk_syms.sh and then mk_syms.sh needs to be executed.

Additional device files to be added need to be added into find_dev.sh. Device option and major and minor device numbers may be discovered by using the command "ls –la" on the device file to see where it is a symlink from. Then use the command "file" which will inform whether it is a character or block device and it's major and minor device number. After the device has been added to the file, find_dev.sh needs to be executed.

First, a generic user jss_user should be created as a local user on the server and clients. The default sandbox created will be stored in jss_user's home directory. This account will allow the clients to connect with the server as this user and also update from jss_user's home directory.

find_lib.sh – This script reads a list of commands from commands_list. It copies the located programs into JSS. The script finds all libraries associated with the command and places those libraries inside lib_file. Once all the associated libraries have been detected, it sorts lib_file alphabetically and then does a uniq, which removes and duplicate libraries.

The first thing to check in the top line of the script is to make sure the path to BASH is correct. For the Ultra V's, the path is /bin/bash. Next replace:
JSS_DIR and DEST: path to where you want the "original default filesystem" to be placed. This will be where you placed jss_user's home directory such as /home/jss_user/root where root will become / after the main program runs the chmod command.
CMD_FILE: Path to file which lists commands
FILE_LIST: Path to file containing specific directories and libraries
Also, the commands defined in the beginning of this script should be checked to ensure they have the correct paths on the machine. This can normally be accomplished with the "which <command>" or a "find <search start path> -name '<command>' –print".

commands_list – Input file for find_lib.sh. This is a user created file which is a compilation of all the programs that will be placed into JSS. They may be modified as the user wishes with one command per line.

lib_file – Output file generated by find_lib.sh. This is a generated file that is deleted and recreated each time this script is executed. This file holds all the libraries that are associated with the commands which are copied into JSS.

find_dir.sh – This script runs identically to the above script but instead of searching for and copying commands into the sandbox, it copies the specific directories and files listed in dir_file_list into JSS. As above, the variables defined at the top of the script should be checked to ensure they are the correct paths. The last command in this script creates the directory var/tmp in JSS. This directory is being created and not copied due to the fact it holds temporary items when programs are being installed on execute on the machine, but is otherwise empty or holds unnecessary data.

dir_file_list – Input file for find_dir.sh. This is a user created file which specifies directories and libraries to be copied into JSS. Directories and libraries may be added into this file (in any order) with one complete directory or library path per line.

find_dev.sh – This program creates special files for character and block devices. Character and block devices are discussed in 5.2. The variables at the top of the script should be checked to ensure they have the correct paths. The script then calls the mknod command which asks for a name for the file to be created, device option (either c for character or b for block), and the major and minor device numbers associated with the device.

find_dot.sh – Due to the algorithm discussed in 5.2 used in copying commands, files and directories into JSS, dot files and compressed files were not being copied correctly. The algorithm was modified in this script to correct this problem. The variables in the top of the script should be checked to ensure they have the correct paths. The script reads in a list of the dot and compressed files and using the altered algorithm copies them into the sandbox.

dot_find – Input file for find_dot.sh. This is a user created file which contains the dot files and compressed files to be copies into JSS. Files may be added with one file per line. Regular files (non dot files or non compressed files) may not be placed into this file or will be copied incorrectly.

mk_syms.sh – This files creates symlinks inside JSS which are located inside the actual filesystem. Symlinks for JSS may only be created going into JSS, or to and from programs located in JSS. There should be **NO** symlinks from JSS to the outside filesystem. The variables in the beginning of this script should be checked to ensure they have the correct paths.

Startup.sh – This is the controlling program to create the original default sandbox. It creates the sandbox directory on the initial call of the script. For updates, it removes and creates the sandbox directory. It calls all the above scripts. Ensure the defines variables have the correct paths to where the scripts are located.

CLIENT - Each time the program is run on the client, it will rsync the client JSS environment from the default sandbox on the server. Then a shell is opened for the user to work in. The user types “exit” to quit out of the environment and close the program. There is no need to make updates to this program except for the additional functionality which is discussed in the section of possible future improvements. The client program needs to be initially compiled and executed on the client machines by a system administrator so they may change the executable of the program on the machine to be SUID root.

Only the program executable will be placed on the clients.

The defines at the top of the script should have the correct paths.

RSYNC_SERVER: username@servername: (path to original default filesystem on the server)
RSYCN_CLIENT: path to JSS on the client machines.
RSYNC_PATH: path to program RSYNC on the client machine
JSS_DIR: path to root directory of JSS
BASH: path to BASH program
LOCK_ME: path of lockfile for JSS which enables only one user to use the program at a time.

First the program checks to see if the lock file (LOCK_ME) for the program has been created. This enables only one user to use the program at a time. If two users launch the program then they will be cast into the same environment where they could affect each other's session. Should the system administrator feel the need to terminate this program, this program should be killed using the "kill –s SIGINT <pid>" command which will close the user's shell and shut down the rest of the program. The program then proceeds to connect to the server to rsync the sandbox environment with the default original sandbox located on the server. Executing rsync consisted of calling the command followed by the options: links, verbose, checksum, recursive, delete, delete-excluded, progress, rsync-path, perms, force and delete. Further explanations of these options can be found in Appendix F. Next the entire JSS environment is chowned to jss_user except for the proc directory, the files .name_service_door (which is the door to nscd) and the password file. The .name_service_door file cannot be copied, only moved and deleted. Should the file .name_service_door get removed, it is recreated by restarting the daemon /usr/sbin/nscd. Next the /proc directory is additionally mounted in JSS. This alleviates the need to unmount /proc, delete /proc and recreate it as a symlink pointing to the new /proc located in JSS. Mounting /proc in JSS is necessary because the original /proc mounted outside the sandbox will not be recognized once the program executes the chroot command. The program then sets the UID and GID of the program to be jss_user and chroots. Once this has been completed, a user shell is launched and they are now contained in JSS. When the user has completed their session, they type "exit" the UID and GID of the program is returned to root. Then the program will umount /proc from JSS, move .name_service_door back to its original location, remove the lock file and finally exit the program.

Should the program unexpectedly get terminated and the lock file not be removed, simply delete the file /var/run/jss.lock, which will remove the lock on the program and allow a user to execute the program.

To get anything using Xwindows to execute, OUTSIDE JSS, the user needs to types the command "xhost +<hostname or ipaddress>", and INSIDE JSS type the command "export DISPLAY=<hostname or ipaddress>:0.0"

SETTING UP RSYNC – For this program, authentication is through an rsync daemon which is run on the server.

SERVER – The is the setup for the server when using the rsync daemon.

Rsyncd.conf – configuration file for the rsync daemon.
[JSS] is the module name.
UID = user ID for the connecting user
GID = for the group ID for the connecting user.
path = specifies the specific path that the user connection will access.
auth users = username of authorized users
secrets file = path to the secrets file

Rsyncd.secrets – file which contains the user ID and password pairs for authorized uses.
User ID : Password

Then the command “rsync --daemon” needs to be started.

CLIENT – This is the setup for the client when using the rsync daemon.

Rsyncd.secrets – file containing user passwords. When using rsync and specifying the option “rsync --password”, the password placed into this file is used when authenticating with the rsync server. There is one user password per line.

Appendix [G] contains sample files used with rsync.

[F] RSYNC Documentation and Checksum Algorithm

WHAT IS RSYNC?
---

rsync is a replacement for rcp that has many more features.

rsync uses the "rsync algorithm" which provides a very fast method for bringing remote files into sync. It does this by sending just the differences in the files across the link, without requiring that both sets of files are present at one of the ends of the link beforehand. At first glance this may seem impossible because the calculation of diffs between two files normally requires local access to both
files.

A technical report describing the rsync algorithm is included with this package.

USAGE

Basically you use rsync just like rcp, but rsync has many additional options.

Here is a brief description of rsync usage:

```
Usage: rsync [OPTION]... SRC [SRC]... [USER@]HOST:DEST
  or   rsync [OPTION]... [USER@]HOST:SRC DEST
  or   rsync [OPTION]... SRC [SRC]... DEST
  or   rsync [OPTION]... [USER@]HOST::SRC [DEST]
  or   rsync [OPTION]... SRC [SRC]... [USER@]HOST::DEST
  or   rsync [OPTION]... rsync://[USER@]HOST[:PORT]/SRC [DEST]
SRC on single-colon remote HOST will be expanded by remote shell
SRC on server remote HOST may contain shell wildcards or multiple
  sources separated by space as long as they have same top-level
```

Options

-v, --verbose	increase verbosity
-q, --quiet	decrease verbosity
-c, --checksum	always checksum
-a, --archive	archive mode
-r, --recursive	recurse into directories
-R, --relative	use relative path names
-b, --backup	make backups (default ~ suffix)
--suffix=SUFFIX	override backup suffix
-u, --update	update only (don't overwrite newer files)
-l, --links	preserve soft links
-L, --copy-links	treat soft links like regular files
--copy-unsafe-links	copy links outside the source tree

```
    --safe-links                ignore links outside the destination tree
 -H, --hard-links               preserve hard links
 -p, --perms                    preserve permissions
 -o, --owner                    preserve owner (root only)
 -g, --group                    preserve group
 -D, --devices                  preserve devices (root only)
 -t, --times                    preserve times
 -S, --sparse                   handle sparse files efficiently
 -n, --dry-run                  show what would have been transferred
 -W, --whole-file               copy whole files, no incremental checks
 -x, --one-file-system          don't cross filesystem boundaries
 -B, --block-size=SIZE          checksum blocking size (default 700)
 -e, --rsh=COMMAND              specify rsh replacement
     --rsync-path=PATH          specify path to rsync on the remote machine
 -C, --cvs-exclude              auto ignore files in the same way CVS does
     --delete                   delete files that don't exist on the sending side
     --delete-excluded          also delete excluded files on the receiving side
     --partial                  keep partially transferred files
     --force                    force deletion of directories even if not empty
     --numeric-ids              don't map uid/gid values by user/group name
     --timeout=TIME             set IO timeout in seconds
 -I, --ignore-times             don't exclude files that match length and time
     --size-only                only use file size when determining if a file should be
transferred
 -T  --temp-dir=DIR             create temporary files in directory DIR
     --compare-dest=DIR         also compare destination files relative to DIR
 -z, --compress                 compress file data
     --exclude=PATTERN          exclude files matching PATTERN
     --exclude-from=FILE        exclude patterns listed in FILE
     --include=PATTERN          don't exclude files matching PATTERN
     --include-from=FILE        don't exclude patterns listed in FILE
     --version                  print version number
     --daemon                   run as a rsync daemon
     --config=FILE              specify alternate rsyncd.conf file
     --port=PORT                specify alternate rsyncd port number
     --stats                    give some file transfer stats
     --progress                 show progress during transfer
     --log-format=FORMAT        log file transfers using specified format
     --password-file=FILE       get password from FILE
 -h, --help                     show this help screen
```

SETUP

Rsync normally uses rsh or ssh for communication. It does not need to be setuid and requires no special privileges for installation. You must, however, have a working rsh or ssh system. Using ssh is recommended for its security features.

Alternatively, rsync can run in `daemon' mode, listening on a socket. This is generally used for public file distribution, although authentication and access control are available.

To install rsync, first run the "configure" script. This will create a makefile and config.h appropriate for your system. Then type "make".

Note that on some systems you will have to force configure not to use gcc because gcc may not support some features (such as 64 bit file offsets) that your system may support. Set the environment variable CC to the name of your native compiler before running configure in this
case.

Once built put a copy of rsync in your search path on the local and remote systems (or use "make install"). That's it!

THE RSYNC ALGORITHM

Suppose we have two general purpose computers α and β. Computer α has access to a file *A* and β has access to file *B*, where *A* and *B* are ``similar". There is a slow communications link between α and β. The rsync algorithm consists of the following steps:

1. β splits the file *B* into a series of non-overlapping fixed-sized blocks of size S bytes[1]. The last block may be shorter than *S* bytes.

2. For each of these blocks β calculates two checksums: a weak ``rolling" 32-bit checksum (described below) and a strong 128-bit MD4 checksum.

3. β sends these checksums to α.

4. α searches through *A* to find all blocks of length *S* bytes (at any offset, not just multiples of *S*) that have the same weak and strong checksum as one of the blocks of *B*. This can be done in a single pass very quickly using a special property of the rolling checksum described below.

5. α sends β a sequence of instructions for constructing a copy of *A*. Each instruction is either a reference to a block of *B*, or literal data. Literal data is sent only for those sections of *A* which did not match any of the blocks of *B*.

The end result is that β gets a copy of *A*, but only the pieces of *A* that are not found in *B* (plus a small amount of data for checksums and block indexes) are sent over the link. The algorithm also only requires one round trip, which minimises the impact of the link latency.
The most important details of the algorithm are the rolling checksum and the associated multi-alternate search mechanism which allows the all-offsets checksum search to proceed very quickly. These will be discussed in greater detail below

ROLLING CHECKSUM

The weak rolling checksum used in the rsync algorithm needs to have the property that it is very cheap to calculate the checksum of a buffer X_2 .. X_{n+1} given the checksum of buffer X_1 .. X_n and the values of the bytes X_1 and X_{n+1}. The weak checksum algorithm we used in our implementation was inspired by Mark Adler's adler-32 checksum. Our checksum is defined by

$$a(k,l) = (\sum_{i=k}^{l} X_i) \bmod M$$

$$b(k,l) = (\sum_{i=k}^{l} (l-i+1)X_i) \bmod M$$

$s(k,l) = a(k,l) + 2^{16}\, b(k,l)$

where $s(k,l)$ is the rolling checksum of the bytes $X_k \ldots X_l$. For simplicity and speed, we use $M = 2^{16}$.
The important property of this checksum is that successive values can be computed very efficiently using the recurrence relations

$$a(k+1,l+1) = (a(k,l) - X_k + X_{l+1}) \bmod M$$

$$b(k+1,l+1) = (b(k,l) - (l-k+1)X_k + a(k+1,l+1)) \bmod M$$

Thus the checksum can be calculated for blocks of length S at all possible offsets within a file in a ``rolling'' fashion, with very little computation at each point.
Despite its simplicity, this checksum was found to be quite adequate as a first level check for a match of two file blocks. We have found in practice that the probability of this checksum matching when the blocks are not equal is quite low. This is important because

the much more expensive strong checksum must be calculated for each block where the weak checksum matches.

CHECKSUM SEARCHING

Once α has received the list of checksums of the blocks of *B*, it must search *A* for any blocks at any offset that match the checksum of some block of *B*. The basic strategy is to compute the 32-bit rolling checksum for a block of length *S* starting at each byte of *A* in turn, and for each checksum, search the list for a match. To do this our implementation uses a simple 3 level searching scheme.

The first level uses a 16-bit hash of the 32-bit rolling checksum and a 2^{16} entry hash table. The list of checksum values (i.e., the checksums from the blocks of *B*) is sorted according to the 16-bit hash of the 32-bit rolling checksum. Each entry in the hash table points to the first element of the list for that hash value, or contains a null value if no element of the list has that hash value.
At each offset in the file the 32-bit rolling checksum and its 16-bit hash are calculated. If the hash table entry for that hash value is not a null value, the second level check is invoked.

The second level check involves scanning the sorted checksum list starting with the entry pointed to by the hash table entry, looking for an entry whose 32-bit rolling checksum matches the current value. The scan terminates when it reaches an entry whose 16-bit hash differs. If this search finds a match, the third level check is invoked.

The third level check involves calculating the strong checksum for the current offset in the file and comparing it with the strong checksum value in the current list entry. If the two strong checksums match, we assume that we have found a block of *A* which matches a block of *B*. In fact the blocks could be different, but the probability of this is microscopic, and in practice this is a reasonable assumption.
When a match is found, α sends β the data in *A* between the current file offset and the end of the previous match, followed by the index of the block in *B* that matched. This data is sent immediately a match is found, which allows us to overlap the communication with further computation.
If no match is found at a given offset in the file, the rolling checksum is updated to the next offset and the search proceeds. If a match is found, the search is restarted at the end of the matched block. This strategy saves a considerable amount of computation for the common case where the two files are nearly identical. In addition, it would be a simple matter to encode the block indexes as runs, for the common case where a portion of *A* matches a series of blocks of *B* in order